

MySQL 5.0 Reference Manual

Copyright 1997-2006 MySQL AB

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms: You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how MySQL disseminates it (that is, electronically for download on a website with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of MySQL AB. MySQL AB reserves any and all rights to this documentation not expressly granted above.

Please email [<docs@mysql.com>](mailto:docs@mysql.com) for more information or if you are interested in doing a translation.

Abstract

This is the MySQL Reference Manual. It documents MySQL 5.0 through 5.0.25.

Document generated on: 2006-08-11

Table of Contents

[Preface](#)

[1. General Information](#)

[1.1. About This Manual](#)

[1.2. Conventions Used in This Manual](#)

[1.3. Overview of MySQL AB](#)

[1.4. Overview of the MySQL Database Management System](#)

[1.4.1. History of MySQL](#)

[1.4.2. The Main Features of MySQL](#)

[1.4.3. MySQL Stability](#)

[1.4.4. How Large MySQL Tables Can Be](#)

[1.4.5. Year 2000 Compliance](#)

[1.5. Overview of the MaxDB Database Management System](#)

[1.5.1. What is MaxDB?](#)

[1.5.2. History of MaxDB](#)

[1.5.3. Features of MaxDB](#)

[1.5.4. Licensing and Support](#)

[1.5.5. Feature Differences Between MaxDB and MySQL](#)

[1.5.6. Interoperability Features Between MaxDB and MySQL](#)

[1.5.7. MaxDB-Related Links](#)

[1.6. MySQL Development Roadmap](#)

[1.6.1. What's New in MySQL 5.0](#)

[1.7. MySQL Information Sources](#)

[1.7.1. MySQL Mailing Lists](#)

[1.7.2. MySQL Community Support at the MySQL Forums](#)

[1.7.3. MySQL Community Support on Internet Relay Chat \(IRC\)](#)

[1.8. How to Report Bugs or Problems](#)

[1.9. MySQL Standards Compliance](#)

[1.9.1. What Standards MySQL Follows](#)

[1.9.2. Selecting SQL Modes](#)

[1.9.3. Running MySQL in ANSI Mode](#)

[1.9.4. MySQL Extensions to Standard SQL](#)

[1.9.5. MySQL Differences from Standard SQL](#)

[1.9.6. How MySQL Deals with Constraints](#)

[2. Installing and Upgrading MySQL](#)

[2.1. General Installation Issues](#)

[2.1.1. Operating Systems Supported by MySQL](#)

[2.1.2. Choosing Which MySQL Distribution to Install](#)

[2.1.3. How to Get MySQL](#)

[2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG](#)

[2.1.5. Installation Layouts](#)

[2.2. Standard MySQL Installation Using a Binary Distribution](#)

[2.3. Installing MySQL on Windows](#)

[2.3.1. Choosing An Installation Package](#)

[2.3.2. Installing MySQL with the Automated Installer](#)

[2.3.3. Using the MySQL Installation Wizard](#)

[2.3.4. Using the Configuration Wizard](#)

[2.3.5. Installing MySQL from a Noinstall Zip Archive](#)

[2.3.6. Extracting the Install Archive](#)

[2.3.7. Creating an Option File](#)

[2.3.8. Selecting a MySQL Server type](#)

[2.3.9. Starting the Server for the First Time](#)

[2.3.10. Starting MySQL from the Windows Command Line](#)

[2.3.11. Starting MySQL as a Windows Service](#)

[2.3.12. Testing The MySQL Installation](#)

[2.3.13. Troubleshooting a MySQL Installation Under Windows](#)

[2.3.14. Upgrading MySQL on Windows](#)

[2.3.15. MySQL on Windows Compared to MySQL on Unix](#)

[2.4. Installing MySQL on Linux](#)

[2.5. Installing MySQL on Mac OS X](#)

[2.6. Installing MySQL on Solaris](#)

[2.7. Installing MySQL on NetWare](#)

[2.8. Installing MySQL on Other Unix-Like Systems](#)

[2.9. MySQL Installation Using a Source Distribution](#)

[2.9.1. Source Installation Overview](#)

[2.9.2. Typical **configure** Options](#)

[2.9.3. Installing from the Development Source Tree](#)

[2.9.4. Dealing with Problems Compiling MySQL](#)

[2.9.5. MIT-pthreads Notes](#)

[2.9.6. Installing MySQL from Source on Windows](#)

[2.9.7. Compiling MySQL Clients on Windows](#)

[2.10. Post-Installation Setup and Testing](#)

- [2.10.1. Windows Post-Installation Procedures](#)
- [2.10.2. Unix Post-Installation Procedures](#)
- [2.10.3. Securing the Initial MySQL Accounts](#)
- [2.11. Upgrading MySQL](#)
 - [2.11.1. Upgrading from MySQL 5.0 to 5.1](#)
 - [2.11.2. Upgrading from MySQL 4.1 to 5.0](#)
 - [2.11.3. Copying MySQL Databases to Another Machine](#)
- [2.12. Downgrading MySQL](#)
 - [2.12.1. Downgrading to MySQL 4.1](#)
- [2.13. Operating System-Specific Notes](#)
 - [2.13.1. Linux Notes](#)
 - [2.13.2. Mac OS X Notes](#)
 - [2.13.3. Solaris Notes](#)
 - [2.13.4. BSD Notes](#)
 - [2.13.5. Other Unix Notes](#)
 - [2.13.6. OS/2 Notes](#)
- [2.14. Perl Installation Notes](#)
 - [2.14.1. Installing Perl on Unix](#)
 - [2.14.2. Installing ActiveState Perl on Windows](#)
 - [2.14.3. Problems Using the Perl DBI/DBD Interface](#)

[3. Tutorial](#)

[3.1. Connecting to and Disconnecting from the Server](#)

[3.2. Entering Queries](#)

[3.3. Creating and Using a Database](#)

[3.3.1. Creating and Selecting a Database](#)

[3.3.2. Creating a Table](#)

[3.3.3. Loading Data into a Table](#)

[3.3.4. Retrieving Information from a Table](#)

[3.4. Getting Information About Databases and Tables](#)

[3.5. Using **mysql** in Batch Mode](#)

[3.6. Examples of Common Queries](#)

[3.6.1. The Maximum Value for a Column](#)

[3.6.2. The Row Holding the Maximum of a Certain Column](#)

[3.6.3. Maximum of Column per Group](#)

[3.6.4. The Rows Holding the Group-wise Maximum of a Certain Field](#)

[3.6.5. Using User-Defined Variables](#)

[3.6.6. Using Foreign Keys](#)

[3.6.7. Searching on Two Keys](#)

[3.6.8. Calculating Visits Per Day](#)

[3.6.9. Using `AUTO INCREMENT`](#)

[3.7. Queries from the Twin Project](#)

[3.7.1. Find All Non-distributed Twins](#)

[3.7.2. Show a Table of Twin Pair Status](#)

[3.8. Using MySQL with Apache](#)

[4. Using MySQL Programs](#)

[4.1. Overview of MySQL Programs](#)

[4.2. Invoking MySQL Programs](#)

[4.3. Specifying Program Options](#)

[4.3.1. Using Options on the Command Line](#)

[4.3.2. Using Option Files](#)

[4.3.3. Using Environment Variables to Specify Options](#)

[4.3.4. Using Options to Set Program Variables](#)

[5. Database Administration](#)

[5.1. Overview of Server-Side Programs](#)

[5.2. **mysqld** — The MySQL Server](#)

[5.2.1. **mysqld** Command Options](#)

[5.2.2. Server System Variables](#)

[5.2.3. Using System Variables](#)

[5.2.4. Server Status Variables](#)

[5.2.5. The Server SQL Mode](#)

[5.2.6. The MySQL Server Shutdown Process](#)

[5.2.7. MySQL Server-Side Help Support](#)

[5.3. The **mysqld-max** Extended MySQL Server](#)

[5.4. MySQL Server Startup Programs](#)

[5.4.1. **mysqld safe** — MySQL Server Startup Script](#)

[5.4.2. **mysql.server** — MySQL Server Startup Script](#)

[5.4.3. **mysqld multi** — Manage Multiple MySQL Servers](#)

[5.5. **mysqlmanager** — The MySQL Instance Manager](#)

[5.5.1. Starting the MySQL Server with MySQL Instance Manager](#)

[5.5.2. Connecting to the MySQL Instance Manager and Creating User Accounts](#)

[5.5.3. MySQL Instance Manager Command Options](#)

[5.5.4. MySQL Instance Manager Configuration Files](#)

[5.5.5. Commands Recognized by the MySQL Instance Manager](#)

[5.6. Installation-Related Programs](#)

[5.6.1. **mysql_fix_privilege_tables** — Upgrade MySQL System Tables](#)

[5.6.2. **mysql_upgrade** — Check Tables for MySQL Upgrade](#)

[5.7. General Security Issues](#)

[5.7.1. General Security Guidelines](#)

[5.7.2. Making MySQL Secure Against Attackers](#)

[5.7.3. Security-Related **mysqld** Options](#)

[5.7.4. Security Issues with `LOAD DATA LOCAL`](#)

[5.7.5. How to Run MySQL as a Normal User](#)

[5.8. The MySQL Access Privilege System](#)

[5.8.1. What the Privilege System Does](#)

[5.8.2. How the Privilege System Works](#)

[5.8.3. Privileges Provided by MySQL](#)

[5.8.4. Connecting to the MySQL Server](#)

[5.8.5. Access Control, Stage 1: Connection Verification](#)

[5.8.6. Access Control, Stage 2: Request Verification](#)

- [5.8.7. When Privilege Changes Take Effect](#)
- [5.8.8. Causes of Access denied Errors](#)
- [5.8.9. Password Hashing as of MySQL 4.1](#)
- [5.9. MySQL User Account Management](#)
 - [5.9.1. MySQL Usernames and Passwords](#)
 - [5.9.2. Adding New User Accounts to MySQL](#)
 - [5.9.3. Removing User Accounts from MySQL](#)
 - [5.9.4. Limiting Account Resources](#)
 - [5.9.5. Assigning Account Passwords](#)
 - [5.9.6. Keeping Your Password Secure](#)
 - [5.9.7. Using Secure Connections](#)
- [5.10. Backup and Recovery](#)
 - [5.10.1. Database Backups](#)
 - [5.10.2. Example Backup and Recovery Strategy](#)
 - [5.10.3. Point-in-Time Recovery](#)
 - [5.10.4. Table Maintenance and Crash Recovery](#)
- [5.11. MySQL Localization and International Usage](#)
 - [5.11.1. The Character Set Used for Data and Sorting](#)
 - [5.11.2. Setting the Error Message Language](#)
 - [5.11.3. Adding a New Character Set](#)
 - [5.11.4. The Character Definition Arrays](#)
 - [5.11.5. String Collating Support](#)
 - [5.11.6. Multi-Byte Character Support](#)
 - [5.11.7. Problems With Character Sets](#)
 - [5.11.8. MySQL Server Time Zone Support](#)
- [5.12. MySQL Server Logs](#)
 - [5.12.1. The Error Log](#)
 - [5.12.2. The General Query Log](#)
 - [5.12.3. The Binary Log](#)
 - [5.12.4. The Slow Query Log](#)
 - [5.12.5. Server Log Maintenance](#)
- [5.13. Running Multiple MySQL Servers on the Same Machine](#)
 - [5.13.1. Running Multiple Servers on Windows](#)
 - [5.13.2. Running Multiple Servers on Unix](#)
 - [5.13.3. Using Client Programs in a Multiple-Server Environment](#)
- [5.14. The MySQL Query Cache](#)
 - [5.14.1. How the Query Cache Operates](#)
 - [5.14.2. Query Cache SELECT Options](#)
 - [5.14.3. Query Cache Configuration](#)

[5.14.4. Query Cache Status and Maintenance](#)

[6. Replication](#)

[6.1. Introduction to Replication](#)

[6.2. Replication Implementation Overview](#)

[6.3. Replication Implementation Details](#)

[6.3.1. Replication Master Thread States](#)

[6.3.2. Replication Slave I/O Thread States](#)

[6.3.3. Replication Slave SQL Thread States](#)

[6.3.4. Replication Relay and Status Files](#)

[6.4. How to Set Up Replication](#)

[6.5. Replication Compatibility Between MySQL Versions](#)

[6.6. Upgrading a Replication Setup](#)

[6.6.1. Upgrading Replication to 5.0](#)

[6.7. Replication Features and Known Problems](#)

[6.8. Replication Startup Options](#)

[6.9. How Servers Evaluate Replication Rules](#)

[6.10. Replication FAQ](#)

[6.11. Troubleshooting Replication](#)

[6.12. How to Report Replication Bugs or Problems](#)

[6.13. Auto-Increment in Multiple-Master Replication](#)

[7. Optimization](#)

[7.1. Optimization Overview](#)

[7.1.1. MySQL Design Limitations and Tradeoffs](#)

[7.1.2. Designing Applications for Portability](#)

[7.1.3. What We Have Used MySQL For](#)

[7.1.4. The MySQL Benchmark Suite](#)

[7.1.5. Using Your Own Benchmarks](#)

[7.2. Optimizing SELECT and Other Statements](#)

[7.2.1. Optimizing Queries with EXPLAIN](#)

[7.2.2. Estimating Query Performance](#)

[7.2.3. Speed of SELECT Queries](#)

[7.2.4. WHERE Clause Optimization](#)

[7.2.5. Range Optimization](#)

[7.2.6. Index Merge Optimization](#)

[7.2.7. IS NULL Optimization](#)

[7.2.8. DISTINCT Optimization](#)

[7.2.9. LEFT JOIN and RIGHT JOIN Optimization](#)

[7.2.10. Nested Join Optimization](#)

[7.2.11. Outer Join Simplification](#)

[7.2.12. ORDER BY Optimization](#)

[7.2.13. GROUP BY Optimization](#)

[7.2.14. LIMIT Optimization](#)

[7.2.15. How to Avoid Table Scans](#)

[7.2.16. Speed of INSERT Statements](#)

[7.2.17. Speed of UPDATE Statements](#)

[7.2.18. Speed of DELETE Statements](#)

[7.2.19. Other Optimization Tips](#)

[7.3. Locking Issues](#)

[7.3.1. Locking Methods](#)

[7.3.2. Table Locking Issues](#)

[7.3.3. Concurrent Inserts](#)

[7.4. Optimizing Database Structure](#)

[7.4.1. Design Choices](#)

[7.4.2. Make Your Data as Small as Possible](#)

[7.4.3. Column Indexes](#)

[7.4.4. Multiple-Column Indexes](#)

[7.4.5. How MySQL Uses Indexes](#)

[7.4.6. The MyISAM Key Cache](#)

[7.4.7. MyISAM Index Statistics Collection](#)

[7.4.8. How MySQL Opens and Closes Tables](#)

[7.4.9. Drawbacks to Creating Many Tables in the Same Database](#)

[7.5. Optimizing the MySQL Server](#)

[7.5.1. System Factors and Startup Parameter Tuning](#)

[7.5.2. Tuning Server Parameters](#)

[7.5.3. Controlling Query Optimizer Performance](#)

[7.5.4. How Compiling and Linking Affects the Speed of MySQL](#)

[7.5.5. How MySQL Uses Memory](#)

[7.5.6. How MySQL Uses DNS](#)

[7.6. Disk Issues](#)

[7.6.1. Using Symbolic Links](#)

[8. Client and Utility Programs](#)

[8.1. Overview of Client and Utility Programs](#)

[8.2. **mysam_ftdump** — Display Full-Text Index information](#)

[8.3. **mysamchk** — MyISAM Table-Maintenance Utility](#)

[8.3.1. **mysamchk** General Options](#)

[8.3.2. **mysamchk** Check Options](#)

[8.3.3. **mysamchk** Repair Options](#)

[8.3.4. Other **mysamchk** Options](#)

[8.3.5. **mysamchk** Memory Usage](#)

[8.4. **mysamlog** — Display MyISAM Log File Contents](#)

[8.5. **mysampack** — Generate Compressed, Read-Only MyISAM Tables](#)

[8.6. **mysql** — The MySQL Command-Line Tool](#)

[8.6.1. **mysql** Options](#)

[8.6.2. **mysql** Commands](#)

[8.6.3. **mysql** Server-Side Help](#)

[8.6.4. Executing SQL Statements from a Text File](#)

[8.6.5. **mysql** Tips](#)

[8.7. **mysql_explain_log** — Use EXPLAIN on Statements in Query Log](#)

[8.8. **mysqlaccess** — Client for Checking Access Privileges](#)

[8.9. **mysqladmin** — Client for Administering a MySQL Server](#)

[8.10. **mysqlbinlog** — Utility for Processing Binary Log Files](#)

[8.11. **mysqlcheck** — A Table Maintenance and Repair Program](#)

[8.12. **mysqldump** — A Database Backup Program](#)

[8.13. **mysqlhotcopy** — A Database Backup Program](#)

[8.14. **mysqlimport** — A Data Import Program](#)

[8.15. **mysqlshow** — Display Database, Table, and Column Information](#)

[8.16. **mysql_zap** — Kill Processes That Match a Pattern](#)

[8.17. **perror** — Explain Error Codes](#)

[8.18. **replace** — A String-Replacement Utility](#)

9. Language Structure

9.1. Literal Values

9.1.1. Strings

9.1.2. Numbers

9.1.3. Hexadecimal Values

9.1.4. Boolean Values

9.1.5. Bit-Field Values

9.1.6. NULL Values

9.2. Database, Table, Index, Column, and Alias Names

9.2.1. Identifier Qualifiers

9.2.2. Identifier Case Sensitivity

9.3. User-Defined Variables

9.4. Comment Syntax

9.5. Treatment of Reserved Words in MySQL

[10. Character Set Support](#)

[10.1. Character Sets and Collations in General](#)

[10.2. Character Sets and Collations in MySQL](#)

[10.3. Specifying Character Sets and Collations](#)

[10.3.1. Server Character Set and Collation](#)

[10.3.2. Database Character Set and Collation](#)

[10.3.3. Table Character Set and Collation](#)

[10.3.4. Column Character Set and Collation](#)

[10.3.5. Character String Literal Character Set and Collation](#)

[10.3.6. National Character Set](#)

[10.3.7. Examples of Character Set and Collation Assignment](#)

[10.3.8. Compatibility with Other DBMSs](#)

[10.4. Connection Character Sets and Collations](#)

[10.5. Collation Issues](#)

[10.5.1. Using COLLATE in SQL Statements](#)

[10.5.2. COLLATE Clause Precedence](#)

[10.5.3. BINARY Operator](#)

[10.5.4. Some Special Cases Where the Collation Determination Is Tricky](#)

[10.5.5. Collations Must Be for the Right Character Set](#)

[10.5.6. An Example of the Effect of Collation](#)

[10.6. Operations Affected by Character Set Support](#)

[10.6.1. Result Strings](#)

[10.6.2. CONVERT\(\) and CAST\(\)](#)

[10.6.3. SHOW Statements and INFORMATION_SCHEMA](#)

[10.7. Unicode Support](#)

[10.8. UTF-8 for Metadata](#)

[10.9. Character Sets and Collations That MySQL Supports](#)

[10.9.1. Unicode Character Sets](#)

[10.9.2. West European Character Sets](#)

[10.9.3. Central European Character Sets](#)

[10.9.4. South European and Middle East Character Sets](#)

[10.9.5. Baltic Character Sets](#)

[10.9.6. Cyrillic Character Sets](#)

[10.9.7. Asian Character Sets](#)

[10.10. FAQ: MySQL Chinese, Japanese, and Korean Character Sets](#)

[10.10.1. SELECT shows non-Latin characters as "?"s. Why?](#)

[10.10.2. Troubles with GB character sets \(Chinese\)](#)

- [10.10.3. Troubles with big5 character set \(Chinese\)](#)
- [10.10.4. Troubles with character-set conversions \(Japanese\)](#)
- [10.10.5. The Great Yen Sign problem \(Japanese\)](#)
- [10.10.6. Troubles with euckr character set \(Korean\)](#)
- [10.10.7. The “Data truncated” message](#)
- [10.10.8. Troubles with Access, Perl, PHP, etc.](#)
- [10.10.9. How can I get old MySQL 4.0 behaviour back?](#)
- [10.10.10. Why do some LIKE and FULLTEXT searches fail?](#)
- [10.10.11. What CJK character sets are available?](#)
- [10.10.12. Is character X available in all character sets?](#)
- [10.10.13. Strings don't sort correctly in Unicode \(I\)](#)
- [10.10.14. Strings don't sort correctly in Unicode \(II\)](#)
- [10.10.15. My supplementary characters get rejected](#)
- [10.10.16. Shouldn't it be CJKV \(V for Vietnamese\)?](#)
- [10.10.17. Will MySQL fix any CJK problems in version 5.1?](#)
- [10.10.18. When will MySQL translate the manual again?](#)
- [10.10.19. Whom can I talk to?](#)

[11. Data Types](#)

[11.1. Data Type Overview](#)

[11.1.1. Overview of Numeric Types](#)

[11.1.2. Overview of Date and Time Types](#)

[11.1.3. Overview of String Types](#)

[11.1.4. Data Type Default Values](#)

[11.2. Numeric Types](#)

[11.3. Date and Time Types](#)

[11.3.1. The DATETIME, DATE, and TIMESTAMP Types](#)

[11.3.2. The TIME Type](#)

[11.3.3. The YEAR Type](#)

[11.3.4. Y2K Issues and Date Types](#)

[11.4. String Types](#)

[11.4.1. The CHAR and VARCHAR Types](#)

[11.4.2. The BINARY and VARBINARY Types](#)

[11.4.3. The BLOB and TEXT Types](#)

[11.4.4. The ENUM Type](#)

[11.4.5. The SET Type](#)

[11.5. Data Type Storage Requirements](#)

[11.6. Choosing the Right Type for a Column](#)

[11.7. Using Data Types from Other Database Engines](#)

[12. Functions and Operators](#)

[12.1. Operators](#)

[12.1.1. Operator Precedence](#)

[12.1.2. Type Conversion in Expression Evaluation](#)

[12.1.3. Comparison Functions and Operators](#)

[12.1.4. Logical Operators](#)

[12.2. Control Flow Functions](#)

[12.3. String Functions](#)

[12.3.1. String Comparison Functions](#)

[12.4. Numeric Functions](#)

[12.4.1. Arithmetic Operators](#)

[12.4.2. Mathematical Functions](#)

[12.5. Date and Time Functions](#)

[12.6. What Calendar Is Used By MySQL?](#)

[12.7. Full-Text Search Functions](#)

[12.7.1. Boolean Full-Text Searches](#)

[12.7.2. Full-Text Searches with Query Expansion](#)

[12.7.3. Full-Text Stopwords](#)

[12.7.4. Full-Text Restrictions](#)

[12.7.5. Fine-Tuning MySQL Full-Text Search](#)

[12.8. Cast Functions and Operators](#)

[12.9. Other Functions](#)

[12.9.1. Bit Functions](#)

[12.9.2. Encryption and Compression Functions](#)

[12.9.3. Information Functions](#)

[12.9.4. Miscellaneous Functions](#)

[12.10. Functions and Modifiers for Use with GROUP BY Clauses](#)

[12.10.1. GROUP BY \(Aggregate\) Functions](#)

[12.10.2. GROUP BY Modifiers](#)

[12.10.3. GROUP BY and HAVING with Hidden Fields](#)

[13. SQL Statement Syntax](#)

[13.1. Data Definition Statements](#)

[13.1.1. ALTER DATABASE Syntax](#)

[13.1.2. ALTER TABLE Syntax](#)

[13.1.3. CREATE DATABASE Syntax](#)

[13.1.4. CREATE INDEX Syntax](#)

[13.1.5. CREATE TABLE Syntax](#)

[13.1.6. DROP DATABASE Syntax](#)

[13.1.7. DROP INDEX Syntax](#)

[13.1.8. DROP TABLE Syntax](#)

[13.1.9. RENAME TABLE Syntax](#)

[13.2. Data Manipulation Statements](#)

[13.2.1. DELETE Syntax](#)

[13.2.2. DO Syntax](#)

[13.2.3. HANDLER Syntax](#)

[13.2.4. INSERT Syntax](#)

[13.2.5. LOAD DATA INFILE Syntax](#)

[13.2.6. REPLACE Syntax](#)

[13.2.7. SELECT Syntax](#)

[13.2.8. Subquery Syntax](#)

[13.2.9. TRUNCATE Syntax](#)

[13.2.10. UPDATE Syntax](#)

[13.3. MySQL Utility Statements](#)

[13.3.1. DESCRIBE Syntax](#)

[13.3.2. HELP Syntax](#)

[13.3.3. USE Syntax](#)

[13.4. MySQL Transactional and Locking Statements](#)

[13.4.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax](#)

[13.4.2. Statements That Cannot Be Rolled Back](#)

[13.4.3. Statements That Cause an Implicit Commit](#)

[13.4.4. SAVEPOINT and ROLLBACK TO SAVEPOINT Syntax](#)

[13.4.5. LOCK TABLES and UNLOCK TABLES Syntax](#)

[13.4.6. SET TRANSACTION Syntax](#)

[13.4.7. XA Transactions](#)

[13.5. Database Administration Statements](#)

[13.5.1. Account Management Statements](#)

[13.5.2. Table Maintenance Statements](#)

[13.5.3. SET Syntax](#)

[13.5.4. SHOW Syntax](#)

[13.5.5. Other Administrative Statements](#)

[13.6. Replication Statements](#)

[13.6.1. SQL Statements for Controlling Master Servers](#)

[13.6.2. SQL Statements for Controlling Slave Servers](#)

[13.7. SQL Syntax for Prepared Statements](#)

[14. Storage Engines and Table Types](#)

[14.1. The MyISAM Storage Engine](#)

[14.1.1. MyISAM Startup Options](#)

[14.1.2. Space Needed for Keys](#)

[14.1.3. MyISAM Table Storage Formats](#)

[14.1.4. MyISAM Table Problems](#)

[14.2. The InnoDB Storage Engine](#)

[14.2.1. InnoDB Overview](#)

[14.2.2. InnoDB Contact Information](#)

[14.2.3. InnoDB Configuration](#)

[14.2.4. InnoDB Startup Options and System Variables](#)

[14.2.5. Creating the InnoDB Tablespace](#)

[14.2.6. Creating and Using InnoDB Tables](#)

[14.2.7. Adding and Removing InnoDB Data and Log Files](#)

[14.2.8. Backing Up and Recovering an InnoDB Database](#)

[14.2.9. Moving an InnoDB Database to Another Machine](#)

[14.2.10. InnoDB Transaction Model and Locking](#)

[14.2.11. InnoDB Performance Tuning Tips](#)

[14.2.12. Implementation of Multi-Versioning](#)

[14.2.13. InnoDB Table and Index Structures](#)

[14.2.14. InnoDB File Space Management and Disk I/O](#)

[14.2.15. InnoDB Error Handling](#)

[14.2.16. Restrictions on InnoDB Tables](#)

[14.2.17. InnoDB Troubleshooting](#)

[14.3. The MERGE Storage Engine](#)

[14.3.1. MERGE Table Problems](#)

[14.4. The MEMORY \(HEAP\) Storage Engine](#)

[14.5. The BDB \(BerkeleyDB\) Storage Engine](#)

[14.5.1. Operating Systems Supported by BDB](#)

[14.5.2. Installing BDB](#)

[14.5.3. BDB Startup Options](#)

[14.5.4. Characteristics of BDB Tables](#)

[14.5.5. Restrictions on BDB Tables](#)

[14.5.6. Errors That May Occur When Using BDB Tables](#)

[14.6. The EXAMPLE Storage Engine](#)

[14.7. The FEDERATED Storage Engine](#)

[14.7.1. Description of the FEDERATED Storage Engine](#)

[14.7.2. How to use FEDERATED Tables](#)

- [14.7.3. Limitations of the FEDERATED Storage Engine](#)
- [14.8. The ARCHIVE Storage Engine](#)
- [14.9. The CSV Storage Engine](#)
- [14.10. The BLACKHOLE Storage Engine](#)

[15. MySQL Cluster](#)

[15.1. MySQL Cluster Overview](#)

[15.2. Basic MySQL Cluster Concepts](#)

[15.2.1. MySQL Cluster Nodes, Node Groups, Replicas, and Partitions](#)

[15.3. Simple Multi-Computer How-To](#)

[15.3.1. Hardware, Software, and Networking](#)

[15.3.2. Multi-Computer Installation](#)

[15.3.3. Multi-Computer Configuration](#)

[15.3.4. Initial Startup](#)

[15.3.5. Loading Sample Data and Performing Queries](#)

[15.3.6. Safe Shutdown and Restart](#)

[15.4. MySQL Cluster Configuration](#)

[15.4.1. Building MySQL Cluster from Source Code](#)

[15.4.2. Installing the Software](#)

[15.4.3. Quick Test Setup of MySQL Cluster](#)

[15.4.4. Configuration File](#)

[15.4.5. Overview of Cluster Configuration Parameters](#)

[15.4.6. Configuring Parameters for Local Checkpoints](#)

[15.5. Upgrading and Downgrading MySQL Cluster](#)

[15.5.1. Performing a Rolling Restart of the Cluster](#)

[15.5.2. Cluster Upgrade and Downgrade Compatibility](#)

[15.6. Process Management in MySQL Cluster](#)

[15.6.1. MySQL Server Process Usage for MySQL Cluster](#)

[15.6.2. **ndbd**, the Storage Engine Node Process](#)

[15.6.3. **ndb_mgmd**, the Management Server Process](#)

[15.6.4. **ndb_mgm**, the Management Client Process](#)

[15.6.5. Command Options for MySQL Cluster Processes](#)

[15.7. Management of MySQL Cluster](#)

[15.7.1. MySQL Cluster Startup Phases](#)

[15.7.2. Commands in the Management Client](#)

[15.7.3. Event Reports Generated in MySQL Cluster](#)

[15.7.4. Single-User Mode](#)

[15.8. On-line Backup of MySQL Cluster](#)

[15.8.1. Cluster Backup Concepts](#)

[15.8.2. Using The Management Client to Create a Backup](#)

[15.8.3. How to Restore a Cluster Backup](#)

[15.8.4. Configuration for Cluster Backup](#)

[15.8.5. Backup Troubleshooting](#)

- [15.9. Using High-Speed Interconnects with MySQL Cluster](#)
 - [15.9.1. Configuring MySQL Cluster to use SCI Sockets](#)
 - [15.9.2. Understanding the Impact of Cluster Interconnects](#)
- [15.10. Known Limitations of MySQL Cluster](#)
- [15.11. MySQL Cluster Development Roadmap](#)
 - [15.11.1. MySQL Cluster Changes in MySQL 5.0](#)
 - [15.11.2. MySQL 5.1 Development Roadmap for MySQL Cluster](#)
- [15.12. MySQL Cluster FAQ](#)
- [15.13. MySQL Cluster Glossary](#)

[16. Spatial Extensions](#)

[16.1. Introduction to MySQL Spatial Support](#)

[16.2. The OpenGIS Geometry Model](#)

[16.2.1. The Geometry Class Hierarchy](#)

[16.2.2. Class Geometry](#)

[16.2.3. Class Point](#)

[16.2.4. Class curve](#)

[16.2.5. Class LineString](#)

[16.2.6. Class Surface](#)

[16.2.7. Class Polygon](#)

[16.2.8. Class GeometryCollection](#)

[16.2.9. Class MultiPoint](#)

[16.2.10. Class MultiCurve](#)

[16.2.11. Class MultiLineString](#)

[16.2.12. Class MultiSurface](#)

[16.2.13. Class MultiPolygon](#)

[16.3. Supported Spatial Data Formats](#)

[16.3.1. Well-Known Text \(WKT\) Format](#)

[16.3.2. Well-Known Binary \(WKB\) Format](#)

[16.4. Creating a Spatially Enabled MySQL Database](#)

[16.4.1. MySQL Spatial Data Types](#)

[16.4.2. Creating Spatial Values](#)

[16.4.3. Creating Spatial Columns](#)

[16.4.4. Populating Spatial Columns](#)

[16.4.5. Fetching Spatial Data](#)

[16.5. Analyzing Spatial Information](#)

[16.5.1. Geometry Format Conversion Functions](#)

[16.5.2. Geometry Functions](#)

[16.5.3. Functions That Create New Geometries from Existing Ones](#)

[16.5.4. Functions for Testing Spatial Relations Between Geometric Objects](#)

[16.5.5. Relations on Geometry Minimal Bounding Rectangles \(MBRs\)](#)

[16.5.6. Functions That Test Spatial Relationships Between Geometries](#)

[16.6. Optimizing Spatial Analysis](#)

[16.6.1. Creating Spatial Indexes](#)

[16.6.2. Using a Spatial Index](#)

[16.7. MySQL Conformance and Compatibility](#)

[17. Stored Procedures and Functions](#)

[17.1. Stored Routines and the Grant Tables](#)

[17.2. Stored Routine Syntax](#)

[17.2.1. CREATE PROCEDURE and CREATE FUNCTION Syntax](#)

[17.2.2. ALTER PROCEDURE and ALTER FUNCTION Syntax](#)

[17.2.3. DROP PROCEDURE and DROP FUNCTION Syntax](#)

[17.2.4. CALL Statement Syntax](#)

[17.2.5. BEGIN . . . END Compound Statement Syntax](#)

[17.2.6. DECLARE Statement Syntax](#)

[17.2.7. Variables in Stored Routines](#)

[17.2.8. Conditions and Handlers](#)

[17.2.9. Cursors](#)

[17.2.10. Flow Control Constructs](#)

[17.3. Stored Procedures, Functions, Triggers, and Replication: Frequently Asked Questions](#)

[17.4. Binary Logging of Stored Routines and Triggers](#)

18. Triggers

18.1. CREATE TRIGGER Syntax

18.2. DROP TRIGGER Syntax

18.3. Using Triggers

19. Views

19.1. ALTER VIEW Syntax

19.2. CREATE VIEW Syntax

19.3. DROP VIEW Syntax

[20. The INFORMATION_SCHEMA Database](#)

[20.1. The INFORMATION_SCHEMA SCHEMATA Table](#)

[20.2. The INFORMATION_SCHEMA TABLES Table](#)

[20.3. The INFORMATION_SCHEMA COLUMNS Table](#)

[20.4. The INFORMATION_SCHEMA STATISTICS Table](#)

[20.5. The INFORMATION_SCHEMA USER_PRIVILEGES Table](#)

[20.6. The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table](#)

[20.7. The INFORMATION_SCHEMA TABLE_PRIVILEGES Table](#)

[20.8. The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table](#)

[20.9. The INFORMATION_SCHEMA CHARACTER_SETS Table](#)

[20.10. The INFORMATION_SCHEMA COLLATIONS Table](#)

[20.11. The INFORMATION_SCHEMA](#)

[COLLATION_CHARACTER_SET_APPLICABILITY Table](#)

[20.12. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table](#)

[20.13. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table](#)

[20.14. The INFORMATION_SCHEMA ROUTINES Table](#)

[20.15. The INFORMATION_SCHEMA VIEWS Table](#)

[20.16. The INFORMATION_SCHEMA TRIGGERS Table](#)

[20.17. Other INFORMATION_SCHEMA Tables](#)

[20.18. Extensions to SHOW Statements](#)

[21. Precision Math](#)

[21.1. Types of Numeric Values](#)

[21.2. DECIMAL Data Type Changes](#)

[21.3. Expression Handling](#)

[21.4. Rounding Behavior](#)

[21.5. Precision Math Examples](#)

[22. APIs and Libraries](#)

[22.1. libmysqld, the Embedded MySQL Server Library](#)

[22.2. MySQL C API](#)

[22.2.1. C API Data types](#)

[22.2.2. C API Function Overview](#)

[22.2.3. C API Function Descriptions](#)

[22.2.4. C API Prepared Statements](#)

[22.2.5. C API Prepared Statement Data types](#)

[22.2.6. C API Prepared Statement Function Overview](#)

[22.2.7. C API Prepared Statement Function Descriptions](#)

[22.2.8. C API Prepared statement problems](#)

[22.2.9. C API Handling of Multiple Statement Execution](#)

[22.2.10. C API Handling of Date and Time Values](#)

[22.2.11. C API Threaded Function Descriptions](#)

[22.2.12. C API Embedded Server Function Descriptions](#)

[22.2.13. Common Questions and Problems When Using the C API](#)

[22.2.14. Building Client Programs](#)

[22.2.15. How to Make a Threaded Client](#)

[22.3. MySQL PHP API](#)

[22.3.1. Common Problems with MySQL and PHP](#)

[22.3.2. Enabling Both `mysql` and `mysql_i` in PHP](#)

[22.4. MySQL Perl API](#)

[22.5. MySQL C++ API](#)

[22.5.1. Borland C++](#)

[22.6. MySQL Python API](#)

[22.7. MySQL Tcl API](#)

[22.8. MySQL Eiffel Wrapper](#)

[22.9. MySQL Program Development Utilities](#)

[22.9.1. `mysql2mysql` — Convert mSQL Programs for Use with MySQL](#)

[22.9.2. `mysql_config` — Get Compile Options for Compiling Clients](#)

[23. Connectors](#)

[23.1. MySQL Connector/ODBC](#)

[23.1.1. Introduction to MyODBC](#)

[23.1.2. How to Install MyODBC](#)

[23.1.3. MyODBC Configuration](#)

[23.1.4. MyODBC Examples](#)

[23.1.5. MyODBC Reference](#)

[23.1.6. MyODBC Notes and Tips](#)

[23.1.7. MyODBC Support](#)

[23.2. Connector/NET](#)

[23.2.1. Connector/NET Versions](#)

[23.2.2. How to install Connector/NET](#)

[23.2.3. Connector/NET Examples](#)

[23.2.4. Connector/NET Reference](#)

[23.2.5. Connector/NET Notes and Tips](#)

[23.2.6. Connector/NET Support](#)

[23.3. MySQL Connector/J](#)

[23.3.1. Connector/J Versions](#)

[23.3.2. Installing Connector/J](#)

[23.3.3. Connector/J Examples](#)

[23.3.4. Connector/J \(JDBC\) Reference](#)

[23.3.5. Connector/J Notes and Tips](#)

[23.3.6. Connector/J Support](#)

[23.4. MySQL Connector/MXJ](#)

[23.4.1. Introduction to Connector/MXJ](#)

[23.4.2. Installing Connector/MXJ](#)

[23.4.3. Connector/MXJ Configuration](#)

[23.4.4. Connector/MXJ Reference](#)

[23.4.5. Connector/MXJ Notes and Tips](#)

[23.4.6. Connector/MXJ Support](#)

[23.5. Connector/PHP](#)

[24. Extending MySQL](#)

[24.1. MySQL Internals](#)

[24.1.1. MySQL Threads](#)

[24.1.2. MySQL Test Suite](#)

[24.2. Adding New Functions to MySQL](#)

[24.2.1. Features of the User-Defined Function Interface](#)

[24.2.2. CREATE FUNCTION Syntax](#)

[24.2.3. DROP FUNCTION Syntax](#)

[24.2.4. Adding a New User-Defined Function](#)

[24.2.5. Adding a New Native Function](#)

[24.3. Adding New Procedures to MySQL](#)

[24.3.1. Procedure Analyse](#)

[24.3.2. Writing a Procedure](#)

[A. Problems and Common Errors](#)

[A.1. How to Determine What Is Causing a Problem](#)

[A.2. Common Errors When Using MySQL Programs](#)

[A.2.1. Access denied](#)

[A.2.2. Can't connect to \[local\] MySQL server](#)

[A.2.3. Client does not support authentication protocol](#)

[A.2.4. Password Fails When Entered Interactively](#)

[A.2.5. Host 'host name' is blocked](#)

[A.2.6. Too many connections](#)

[A.2.7. Out of memory](#)

[A.2.8. MySQL server has gone away](#)

[A.2.9. Packet too large](#)

[A.2.10. Communication Errors and Aborted Connections](#)

[A.2.11. The table is full](#)

[A.2.12. Can't create/write to file](#)

[A.2.13. Commands out of sync](#)

[A.2.14. Ignoring user](#)

[A.2.15. Table 'tbl name' doesn't exist](#)

[A.2.16. Can't initialize character set](#)

[A.2.17. File Not Found](#)

[A.3. Installation-Related Issues](#)

[A.3.1. Problems Linking to the MySQL Client Library](#)

[A.3.2. Problems with File Permissions](#)

[A.4. Administration-Related Issues](#)

[A.4.1. How to Reset the Root Password](#)

- [A.4.2. What to Do If MySQL Keeps Crashing](#)
 - [A.4.3. How MySQL Handles a Full Disk](#)
 - [A.4.4. Where MySQL Stores Temporary Files](#)
 - [A.4.5. How to Protect or Change the MySQL Unix Socket File](#)
 - [A.4.6. Time Zone Problems](#)
- [A.5. Query-Related Issues](#)
 - [A.5.1. Case Sensitivity in Searches](#)
 - [A.5.2. Problems Using DATE Columns](#)
 - [A.5.3. Problems with NULL Values](#)
 - [A.5.4. Problems with Column Aliases](#)
 - [A.5.5. Rollback Failure for Non-Transactional Tables](#)
 - [A.5.6. Deleting Rows from Related Tables](#)
 - [A.5.7. Solving Problems with No Matching Rows](#)
 - [A.5.8. Problems with Floating-Point Comparisons](#)
- [A.6. Optimizer-Related Issues](#)
- [A.7. Table Definition-Related Issues](#)
 - [A.7.1. Problems with ALTER TABLE](#)
 - [A.7.2. How to Change the Order of Columns in a Table](#)
 - [A.7.3. TEMPORARY TABLE Problems](#)
- [A.8. Known Issues in MySQL](#)
 - [A.8.1. Open Issues in MySQL](#)
- [B. Error Codes and Messages](#)
 - [B.1. Server Error Codes and Messages](#)
 - [B.2. Client Error Codes and Messages](#)
- [C. Credits](#)
 - [C.1. Developers at MySQL AB](#)
 - [C.2. Contributors to MySQL](#)
 - [C.3. Documenters and translators](#)
 - [C.4. Libraries used by and included with MySQL](#)
 - [C.5. Packages that support MySQL](#)
 - [C.6. Tools that were used to create MySQL](#)
 - [C.7. Supporters of MySQL](#)
- [D. MySQL Change History](#)
 - [D.1. Changes in release 5.0.x \(Production\)](#)
 - [D.1.1. Changes in release 5.0.25 \(Not yet released\)](#)
 - [D.1.2. Changes in release 5.0.24 \(Not yet released\)](#)
 - [D.1.3. Changes in release 5.0.23 \(Not released\)](#)
 - [D.1.4. Changes in release 5.0.22 \(24 May 2006\)](#)
 - [D.1.5. Changes in release 5.0.21 \(02 May 2006\)](#)

- [D.1.6. Changes in release 5.0.20a \(18 April 2006\)](#)
- [D.1.7. Changes in release 5.0.20 \(31 March 2006\)](#)
- [D.1.8. Changes in release 5.0.19 \(04 March 2006\)](#)
- [D.1.9. Changes in release 5.0.18 \(21 December 2005\)](#)
- [D.1.10. Changes in release 5.0.17 \(14 December 2005\)](#)
- [D.1.11. Changes in release 5.0.16 \(10 November 2005\)](#)
- [D.1.12. Changes in release 5.0.15 \(19 October 2005: Production\)](#)
- [D.1.13. Changes in release 5.0.14 \(Not released\)](#)
- [D.1.14. Changes in release 5.0.13 \(22 September 2005: Release Candidate\)](#)
- [D.1.15. Changes in release 5.0.12 \(02 September 2005\)](#)
- [D.1.16. Changes in release 5.0.11 \(06 August 2005\)](#)
- [D.1.17. Changes in release 5.0.10 \(27 July 2005\)](#)
- [D.1.18. Changes in release 5.0.9 \(15 July 2005\)](#)
- [D.1.19. Changes in release 5.0.8 \(Not released\)](#)
- [D.1.20. Changes in release 5.0.7 \(10 June 2005\)](#)
- [D.1.21. Changes in release 5.0.6 \(26 May 2005\)](#)
- [D.1.22. Changes in release 5.0.5 \(Not released\)](#)
- [D.1.23. Changes in release 5.0.4 \(16 April 2005\)](#)
- [D.1.24. Changes in release 5.0.3 \(23 March 2005: Beta\)](#)
- [D.1.25. Changes in release 5.0.2 \(01 December 2004\)](#)
- [D.1.26. Changes in release 5.0.1 \(27 July 2004\)](#)
- [D.1.27. Changes in release 5.0.0 \(22 December 2003: Alpha\)](#)

[D.2. Changes in MySQL Cluster](#)

- [D.2.1. Changes in MySQL Cluster-5.0.7 \(10 June 2005\)](#)
- [D.2.2. Changes in MySQL Cluster-5.0.6 \(26 May 2005\)](#)
- [D.2.3. Changes in MySQL Cluster-5.0.5 \(Not released\)](#)
- [D.2.4. Changes in MySQL Cluster-5.0.4 \(16 April 2005\)](#)
- [D.2.5. Changes in MySQL Cluster-5.0.3 \(23 March 2005: Beta\)](#)
- [D.2.6. Changes in MySQL Cluster-5.0.1 \(27 July 2004\)](#)
- [D.2.7. Changes in MySQL Cluster-4.1.13 \(15 July 2005\)](#)
- [D.2.8. Changes in MySQL Cluster-4.1.12 \(13 May 2005\)](#)
- [D.2.9. Changes in MySQL Cluster-4.1.11 \(01 April 2005\)](#)
- [D.2.10. Changes in MySQL Cluster-4.1.10 \(12 February 2005\)](#)
- [D.2.11. Changes in MySQL Cluster-4.1.9 \(13 January 2005\)](#)
- [D.2.12. Changes in MySQL Cluster-4.1.8 \(14 December 2004\)](#)
- [D.2.13. Changes in MySQL Cluster-4.1.7 \(23 October 2004\)](#)
- [D.2.14. Changes in MySQL Cluster-4.1.6 \(10 October 2004\)](#)
- [D.2.15. Changes in MySQL Cluster-4.1.5 \(16 September 2004\)](#)

[D.2.16. Changes in MySQL Cluster-4.1.4 \(31 August 2004\)](#)

[D.2.17. Changes in MySQL Cluster-4.1.3 \(28 June 2004\)](#)

[D.3. MySQL Connector/ODBC \(MyODBC\) Change History](#)

[D.3.1. Changes in MyODBC 3.51.13](#)

[D.3.2. Changes in MyODBC 3.51.12](#)

[D.3.3. Changes in MyODBC 3.51.11](#)

[D.4. MySQL Connector/NET Change History](#)

[D.4.1. Version 1.0.8](#)

[D.4.2. Version 1.0.7](#)

[D.4.3. Version 1.0.6](#)

[D.4.4. Version 1.0.5](#)

[D.4.5. Version 1.0.4 1-20-05](#)

[D.4.6. Version 1.0.3-gamma 12-10-04](#)

[D.4.7. Version 1.0.2-gamma 04-11-15](#)

[D.4.8. Version 1.0.1-beta2 04-10-27](#)

[D.4.9. Version 1.0.0 04-09-01](#)

[D.4.10. Version 0.9.0 04-08-30](#)

[D.4.11. Version 0.76](#)

[D.4.12. Version 0.75](#)

[D.4.13. Version 0.74](#)

[D.4.14. Version 0.71](#)

[D.4.15. Version 0.70](#)

[D.4.16. Version 0.68](#)

[D.4.17. Version 0.65](#)

[D.4.18. Version 0.60](#)

[D.4.19. Version 0.50](#)

[D.5. MySQL Connector/J Change History](#)

[D.5.1. Changes in MySQL Connector/J 5.0.2-beta \(11 July 2006\)](#)

[D.5.2. Changes in MySQL Connector/J 5.0.1-beta \(Not Released\)](#)

[D.5.3. Changes in MySQL Connector/J 5.0.0-beta \(22 December 2005\)](#)

[D.5.4. Changes in MySQL Connector/J 3.1.14 \(not yet released\)](#)

[D.5.5. Changes in MySQL Connector/J 3.1.13 \(26 May 2006\)](#)

[D.5.6. Changes in MySQL Connector/J 3.1.12 \(30 November 2005\)](#)

[D.5.7. Changes in MySQL Connector/J 3.1.11-stable \(07 October 2005\)](#)

[D.5.8. Changes in MySQL Connector/J 3.1.10-stable \(23 June 2005\)](#)

[D.5.9. Changes in MySQL Connector/J 3.1.9-stable \(22 June 2005\)](#)

[D.5.10. Changes in MySQL Connector/J 3.1.8-stable \(14 April 2005\)](#)

[D.5.11. Changes in MySQL Connector/J 3.1.7-stable \(18 February 2005\)](#)

[D.5.12. Changes in MySQL Connector/J 3.1.6-stable \(23 December 2004\)](#)

[D.5.13. Changes in MySQL Connector/J 3.1.5-gamma \(02 December 2004\)](#)

[D.5.14. Changes in MySQL Connector/J 3.1.4-beta \(04 September 2004\)](#)

[D.5.15. Changes in MySQL Connector/J 3.1.3-beta \(07 July 2004\)](#)

[D.5.16. Changes in MySQL Connector/J 3.1.2-alpha \(09 June 2004\)](#)

[D.5.17. Changes in MySQL Connector/J 3.1.1-alpha \(14 February 2004\)](#)

[D.5.18. Changes in MySQL Connector/J 3.1.0-alpha \(18 February 2003\)](#)

[D.5.19. Changes in MySQL Connector/J 3.0.17-ga \(23 June 2005\)](#)

[D.5.20. Changes in MySQL Connector/J 3.0.16-ga \(15 November 2004\)](#)

[D.5.21. Changes in MySQL Connector/J 3.0.15-production \(04 September 2004\)](#)

[D.5.22. Changes in MySQL Connector/J 3.0.14-production \(28 May 2004\)](#)

[D.5.23. Changes in MySQL Connector/J 3.0.13-production \(27 May 2004\)](#)

[D.5.24. Changes in MySQL Connector/J 3.0.12-production \(18 May 2004\)](#)

[D.5.25. Changes in MySQL Connector/J 3.0.11-stable \(19 February 2004\)](#)

[D.5.26. Changes in MySQL Connector/J 3.0.10-stable \(13 January 2004\)](#)

[D.5.27. Changes in MySQL Connector/J 3.0.9-stable \(07 October 2003\)](#)

[D.5.28. Changes in MySQL Connector/J 3.0.8-stable \(23 May 2003\)](#)

[D.5.29. Changes in MySQL Connector/J 3.0.7-stable \(08 April 2003\)](#)

[D.5.30. Changes in MySQL Connector/J 3.0.6-stable \(18 February 2003\)](#)

[D.5.31. Changes in MySQL Connector/J 3.0.5-gamma \(22 January 2003\)](#)

[D.5.32. Changes in MySQL Connector/J 3.0.4-gamma \(06 January 2003\)](#)

[D.5.33. Changes in MySQL Connector/J 3.0.3-dev \(17 December 2002\)](#)
[D.5.34. Changes in MySQL Connector/J 3.0.2-dev \(08 November 2002\)](#)
[D.5.35. Changes in MySQL Connector/J 3.0.1-dev \(21 September 2002\)](#)
[D.5.36. Changes in MySQL Connector/J 3.0.0-dev \(31 July 2002\)](#)
[D.5.37. Changes in MySQL Connector/J 2.0.14 \(16 May 2002\)](#)
[D.5.38. Changes in MySQL Connector/J 2.0.13 \(24 April 2002\)](#)
[D.5.39. Changes in MySQL Connector/J 2.0.12 \(07 April 2002\)](#)
[D.5.40. Changes in MySQL Connector/J 2.0.11 \(27 January 2002\)](#)
[D.5.41. Changes in MySQL Connector/J 2.0.10 \(24 January 2002\)](#)
[D.5.42. Changes in MySQL Connector/J 2.0.9 \(13 January 2002\)](#)
[D.5.43. Changes in MySQL Connector/J 2.0.8 \(25 November 2001\)](#)
[D.5.44. Changes in MySQL Connector/J 2.0.7 \(24 October 2001\)](#)
[D.5.45. Changes in MySQL Connector/J 2.0.6 \(16 June 2001\)](#)
[D.5.46. Changes in MySQL Connector/J 2.0.5 \(13 June 2001\)](#)
[D.5.47. Changes in MySQL Connector/J 2.0.3 \(03 December 2000\)](#)
[D.5.48. Changes in MySQL Connector/J 2.0.1 \(06 April 2000\)](#)
[D.5.49. Changes in MySQL Connector/J 2.0.0pre5 \(21 February 2000\)](#)
[D.5.50. Changes in MySQL Connector/J 2.0.0pre4 \(10 January 2000\)](#)
[D.5.51. Changes in MySQL Connector/J 2.0.0pre \(17 August 1999\)](#)
[D.5.52. Changes in MySQL Connector/J 1.2b \(04 July 1999\)](#)
[D.5.53. Changes in MySQL Connector/J 1.2a \(14 April 1999\)](#)
[D.5.54. Changes in MySQL Connector/J 1.1i \(24 March 1999\)](#)
[D.5.55. Changes in MySQL Connector/J 1.1h \(08 March 1999\)](#)
[D.5.56. Changes in MySQL Connector/J 1.1g \(19 February 1999\)](#)
[D.5.57. Changes in MySQL Connector/J 1.1f \(31 December 1998\)](#)
[D.5.58. Changes in MySQL Connector/J 1.1b \(03 November 1998\)](#)
[D.5.59. Changes in MySQL Connector/J 1.1 \(02 September 1998\)](#)
[D.5.60. Changes in MySQL Connector/J 1.0 \(24 August 1998\)](#)
[D.5.61. Changes in MySQL Connector/J 0.9d \(04 August 1998\)](#)
[D.5.62. Changes in MySQL Connector/J 0.9 \(28 July 1998\)](#)
[D.5.63. Changes in MySQL Connector/J 0.8 \(06 July 1998\)](#)
[D.5.64. Changes in MySQL Connector/J 0.7 \(01 July 1998\)](#)
[D.5.65. Changes in MySQL Connector/J 0.6 \(21 May 1998\)](#)

[E. Porting to Other Systems](#)

[E.1. Debugging a MySQL Server](#)

[E.1.1. Compiling MySQL for Debugging](#)

- [E.1.2. Creating Trace Files](#)
- [E.1.3. Debugging **mysqld** under **gdb**](#)
- [E.1.4. Using a Stack Trace](#)
- [E.1.5. Using Server Logs to Find Causes of Errors in **mysqld**](#)
- [E.1.6. Making a Test Case If You Experience Table Corruption](#)
- [E.2. Debugging a MySQL Client](#)
- [E.3. The DBUG Package](#)
- [E.4. Comments about RTS Threads](#)
- [E.5. Differences Between Thread Packages](#)
- [F. Environment Variables](#)
- [G. Regular Expressions](#)
- [H. Limits in MySQL](#)
 - [H.1. Limits of Joins](#)
- [I. Feature Restrictions](#)
 - [I.1. Restrictions on Stored Routines and Triggers](#)
 - [I.2. Restrictions on Server-Side Cursors](#)
 - [I.3. Restrictions on Subqueries](#)
 - [I.4. Restrictions on Views](#)
 - [I.5. Restrictions on XA Transactions](#)
- [J. GNU General Public License](#)
- [K. MySQL FLOSS License Exception](#)

List of Tables

- 15.1. [Steps for Cluster rolling restarts — by type](#)

List of Examples

- 23.1. [Obtaining a connection from the DriverManager](#)
- 23.2. [Using `java.sql.Statement` to execute a `SELECT` query](#)
- 23.3. [Stored Procedures](#)
- 23.4. [Using `Connection.prepareStatement\(\)`](#)
- 23.5. [Registering output parameters](#)
- 23.6. [Setting `CallableStatement` input parameters](#)
- 23.7. [Retrieving results and output parameter values](#)
- 23.8. [Retrieving `AUTO INCREMENT` column values using `Statement.getGeneratedKeys\(\)`](#)
- 23.9. [Retrieving `AUTO INCREMENT` column values using `SELECT LAST_INSERT_ID\(\)`](#)
- 23.10. [Retrieving `AUTO INCREMENT` column values in `Updatable ResultSets`](#)

- 23.11. [Using a connection pool with a J2EE application server](#)
- 23.12. [Example of transaction with retry logic](#)

Preface

This is the Reference Manual for the MySQL Database System, version 5.0, up to release 5.0.25. It is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.0 and previous versions. If you are using an earlier release of the MySQL software, please refer to the *MySQL 3.23, 4.0, 4.1 Reference Manual*, which provides coverage of the 3.22, 3.23, 4.0, and 4.1 series of MySQL software releases. Differences between minor versions of MySQL 5.0 are noted in the present text with reference to release numbers (5.0.x).

Chapter 1. General Information

Table of Contents

- [1.1. About This Manual](#)
- [1.2. Conventions Used in This Manual](#)
- [1.3. Overview of MySQL AB](#)
- [1.4. Overview of the MySQL Database Management System](#)
 - [1.4.1. History of MySQL](#)
 - [1.4.2. The Main Features of MySQL](#)
 - [1.4.3. MySQL Stability](#)
 - [1.4.4. How Large MySQL Tables Can Be](#)
 - [1.4.5. Year 2000 Compliance](#)
- [1.5. Overview of the MaxDB Database Management System](#)
 - [1.5.1. What is MaxDB?](#)
 - [1.5.2. History of MaxDB](#)
 - [1.5.3. Features of MaxDB](#)
 - [1.5.4. Licensing and Support](#)
 - [1.5.5. Feature Differences Between MaxDB and MySQL](#)
 - [1.5.6. Interoperability Features Between MaxDB and MySQL](#)
 - [1.5.7. MaxDB-Related Links](#)
- [1.6. MySQL Development Roadmap](#)
 - [1.6.1. What's New in MySQL 5.0](#)
- [1.7. MySQL Information Sources](#)
 - [1.7.1. MySQL Mailing Lists](#)
 - [1.7.2. MySQL Community Support at the MySQL Forums](#)
 - [1.7.3. MySQL Community Support on Internet Relay Chat \(IRC\)](#)
- [1.8. How to Report Bugs or Problems](#)
- [1.9. MySQL Standards Compliance](#)
 - [1.9.1. What Standards MySQL Follows](#)
 - [1.9.2. Selecting SQL Modes](#)
 - [1.9.3. Running MySQL in ANSI Mode](#)
 - [1.9.4. MySQL Extensions to Standard SQL](#)
 - [1.9.5. MySQL Differences from Standard SQL](#)
 - [1.9.6. How MySQL Deals with Constraints](#)

The MySQL® software delivers a very fast, multi-threaded, multi-user, and

robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. MySQL is a registered trademark of MySQL AB.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source product under the terms of the GNU General Public License (<http://www.fsf.org/licenses/>) or can purchase a standard commercial license from MySQL AB. See <http://www.mysql.com/company/legal/licensing/> for more information on our licensing policies.

The following list describes some sections of particular interest in this manual:

- For a discussion about the capabilities of the MySQL Database Server, see [Section 1.4.2, “The Main Features of MySQL”](#).
- For installation instructions, see [Chapter 2, *Installing and Upgrading MySQL*](#). For information about upgrading MySQL, see [Section 2.11, “Upgrading MySQL”](#).
- For information about configuring and administering MySQL Server, see [Chapter 5, *Database Administration*](#).
- For information about setting up replication servers, see [Chapter 6, *Replication*](#).
- For tips on porting the MySQL Database Software to new architectures or operating systems, see [Appendix E, *Porting to Other Systems*](#).
- For a tutorial introduction to the MySQL Database Server, see [Chapter 3, *Tutorial*](#).
- For benchmarking information, see the `sql-bench` benchmarking directory in your MySQL distribution.
- For a history of new features and bugfixes, see [Appendix D, *MySQL Change History*](#).
- For a list of currently known bugs and misfeatures, see [Section A.8, “Known Issues in MySQL”](#).

- For future plans, see [Section 1.6, “MySQL Development Roadmap”](#).
- For a list of all the contributors to this project, see [Appendix C, Credits](#).

Important:

To report errors (often called “bugs”), please use the instructions at [Section 1.8, “How to Report Bugs or Problems”](#).

If you have found a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to <security@mysql.com>.

1.1. About This Manual

This is the Reference Manual for the MySQL Database System, version 5.0, through release 5.0.25. It is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.0 and previous versions. If you are using a version 4.1 release of the MySQL software, please refer to the *MySQL 3.23, 4.0, 4.1 Reference Manual*, which covers the 3.23, 4.0, and 4.1 series of MySQL software releases. Differences between minor versions of MySQL 5.0 are noted in the present text with reference to release numbers (5.0.x).

Because this manual serves as a reference, it does not provide general instruction on SQL or relational database concepts. It also does not teach you how to use your operating system or command-line interpreter.

The MySQL Database Software is under constant development, and the Reference Manual is updated frequently as well. The most recent version of the manual is available online in searchable form at <http://dev.mysql.com/doc/>. Other formats also are available there, including HTML, PDF, and Windows CHM versions.

The Reference Manual source files are written in DocBook XML format. The HTML version and other formats are produced automatically, primarily using the DocBook XSL stylesheets. For information about DocBook, see <http://docbook.org/>

The DocBook XML sources of this manual are available from <http://dev.mysql.com/tech-resources/sources.html>. You can check out a copy of the documentation repository with this command:

```
svn checkout http://svn.mysql.com/svnpublic/mysql/doc/
```

If you have any suggestions concerning additions or corrections to this manual, please send them to the documentation team at [<docs@mysql.com>](mailto:docs@mysql.com).

This manual was originally written by David Axmark and Michael “Monty” Widenius. It is maintained by the MySQL Documentation Team, consisting of Paul DuBois, Stefan Hinz, Mike Hillyer, and Jon Stephens. For the many other

contributors, see [Appendix C, Credits](#).

The copyright to this manual is owned by the Swedish company MySQL AB. MySQL® and the MySQL logo are registered trademarks of MySQL AB. Other trademarks and registered trademarks referred to in this manual are the property of their respective owners, and are used for identification purposes only.

1.2. Conventions Used in This Manual

This manual uses certain typographical conventions:

- Text in this style is used for SQL statements; database, table, and column names; program listings and source code; and environment variables. Example: “To reload the grant tables, use the `FLUSH PRIVILEGES` statement.”
- **Text in this style** indicates input that you type in examples.
- **Text in this style** indicates the names of executable programs and scripts, examples being **mysql** (the MySQL command line client program) and **mysqld** (the MySQL server executable).
- *Text in this style* is used for variable input for which you should substitute a value of your own choosing.
- Filenames and directory names are written like this: “The global `my.cnf` file is located in the `/etc` directory.”
- Character sequences are written like this: “To specify a wildcard, use the ‘%’ character.”
- *Text in this style* is used for emphasis.
- **Text in this style** is used in table headings and to convey especially strong emphasis.

When commands are shown that are meant to be executed from within a particular program, the prompt shown preceding the command indicates which command to use. For example, `shell>` indicates a command that you execute from your login shell, and `mysql>` indicates a statement that you execute from the **mysql** client program:

```
shell> type a shell command here
mysql> type a mysql statement here
```

The “shell” is your command interpreter. On Unix, this is typically a program

such as **sh**, **cs****h**, or **ba****sh**. On Windows, the equivalent program is **command.com** or **cmd.exe**, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses *db_name*, *tbl_name*, and *col_name*. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case sensitive and may be written in any lettercase. This manual uses uppercase.

In syntax descriptions, square brackets (‘[’ and ‘]’) indicate optional words or clauses. For example, in the following statement, **IF EXISTS** is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars (‘|’). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets (‘[’ and ‘]’):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces (‘{’ and ‘}’):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (. . .) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, **INSERT . . . SELECT** is shorthand for the form of **INSERT** statement that is followed by a **SELECT** statement.

An ellipsis can also indicate that the preceding syntax element of a statement

may be repeated. In the following example, multiple *reset_option* values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [,reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the `CC` environment variable and run the **configure** command looks like this in Bourne shell syntax:

```
shell> CC=gcc ./configure
```

If you are using **csh** or **tcsh**, you must issue commands somewhat differently:

```
shell> setenv CC gcc  
shell> ./configure
```

1.3. Overview of MySQL AB

MySQL AB is the company of the MySQL founders and main developers. MySQL AB was originally established in Sweden by David Axmark, Allan Larsson, and Michael “Monty” Widenius.

We are dedicated to developing the MySQL database software and promoting it to new users. MySQL AB owns the copyright to the MySQL source code, the MySQL logo and (registered) trademark, and this manual. See [Section 1.4, “Overview of the MySQL Database Management System”](#).

The MySQL core values show our dedication to MySQL and Open Source.

These core values direct how MySQL AB works with the MySQL server software:

- To be the best and the most widely used database in the world
- To be available and affordable by all
- To be easy to use
- To be continuously improved while remaining fast and safe
- To be fun to use and improve
- To be free from bugs

These are the core values of the company MySQL AB and its employees:

- We subscribe to the Open Source philosophy and support the Open Source community
- We aim to be good citizens
- We prefer partners that share our values and mindset
- We answer email and provide support

- We are a virtual company, networking with others
- We work against software patents

The MySQL Web site (<http://www.mysql.com/>) provides the latest information about MySQL and MySQL AB.

By the way, the “AB” part of the company name is the acronym for the Swedish “aktiebolag,” or “stock company.” It translates to “MySQL, Inc.” In fact, MySQL, Inc. and MySQL GmbH are examples of MySQL AB subsidiaries. They are located in the United States and Germany, respectively.

1.4. Overview of the MySQL Database Management System

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by MySQL AB. MySQL AB is a commercial company, founded by the MySQL developers. It is a second generation Open Source company that unites Open Source values and methodology with a successful business model.

The MySQL Web site (<http://www.mysql.com/>) provides the latest information about MySQL software and MySQL AB.

- MySQL is a database management system.

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

- MySQL is a relational database management system.

A relational database stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility. The SQL part of “MySQL” stands for “Structured Query Language.” SQL is the most common standardized language used to access databases and is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” to mean the current version of the SQL Standard at any time.

- MySQL software is Open Source.

Open Source means that it is possible for anyone to use and modify the

software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MySQL software uses the GPL (GNU General Public License), <http://www.fsf.org/licenses/>, to define what you may and may not do with the software in different situations. If you feel uncomfortable with the GPL or need to embed MySQL code into a commercial application, you can buy a commercially licensed version from us. See the MySQL Licensing Overview for more information (<http://www.mysql.com/company/legal/licensing/>).

- The MySQL Database Server is very fast, reliable, and easy to use.

If that is what you are looking for, you should give it a try. MySQL Server also has a practical set of features developed in close cooperation with our users. You can find a performance comparison of MySQL Server with other database managers on our benchmark page. See [Section 7.1.4, “The MySQL Benchmark Suite”](#).

MySQL Server was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Although under constant development, MySQL Server today offers a rich and useful set of functions. Its connectivity, speed, and security make MySQL Server highly suited for accessing databases on the Internet.

- MySQL Server works in client/server or embedded systems.

The MySQL Database Software is a client/server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs).

We also provide MySQL Server as an embedded multi-threaded library that you can link into your application to get a smaller, faster, easier-to-manage standalone product.

- A large amount of contributed MySQL software is available.

It is very likely that your favorite application or language supports the MySQL Database Server.

The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we don't mind if you pronounce it as “my sequel” or in some other localized way.

1.4.1. History of MySQL

We started out with the intention of using the `mSQL` database system to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing, we came to the conclusion that `mSQL` was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as `mSQL`. This API was designed to allow third-party code that was written for use with `mSQL` to be ported easily for use with MySQL.

The derivation of the name MySQL is not clear. Our base directory and a large number of our libraries and tools have had the prefix “my” for well over 10 years. However, co-founder Monty Widenius's daughter is also named My. Which of the two gave its name to MySQL is still a mystery, even for us.

The name of the MySQL Dolphin (our logo) is “Sakila,” which was chosen by the founders of MySQL AB from a huge list of names suggested by users in our “Name the Dolphin” contest. The winning name was submitted by Ambrose Twebaze, an Open Source software developer from Swaziland, Africa. According to Ambrose, the feminine name Sakila has its roots in SiSwati, the local language of Swaziland. Sakila is also the name of a town in Arusha, Tanzania, near Ambrose's country of origin, Uganda.

1.4.2. The Main Features of MySQL

The following list describes some of the important characteristics of the MySQL Database Software. See also [Section 1.6, “MySQL Development Roadmap”](#), for more information about current and upcoming features.

Internals and Portability:

- Written in C and C++.
- Tested with a broad range of different compilers.

- Works on many different platforms. See [Section 2.1.1, “Operating Systems Supported by MySQL”](#).
- Uses GNU Automake, Autoconf, and Libtool for portability.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available. See [Chapter 22, APIs and Libraries](#).
- Fully multi-threaded using kernel threads. It can easily use multiple CPUs if they are available.
- Provides transactional and non-transactional storage engines.
- Uses very fast B-tree disk tables (MyISAM) with index compression.
- Relatively easy to add other storage engines. This is useful if you want to add an SQL interface to an in-house database.
- A very fast thread-based memory allocation system.
- Very fast joins using an optimized one-sweep multi-join.
- In-memory hash tables, which are used as temporary tables.
- SQL functions are implemented using a highly optimized class library and should be as fast as possible. Usually there is no memory allocation at all after query initialization.
- The MySQL code is tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (<http://developer.kde.org/~sewardj/>).
- The server is available as a separate program for use in a client/server networked environment. It is also available as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

Data Types:

- Many data types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long,

FLOAT, DOUBLE, CHAR, VARCHAR, TEXT, BLOB, DATE, TIME, DATETIME, TIMESTAMP, YEAR, SET, ENUM, and OpenGIS spatial types. See [Chapter 11, Data Types](#).

- Fixed-length and variable-length records.

Statements and Functions:

- Full operator and function support in the SELECT and WHERE clauses of queries. For example:

```
mysql> SELECT CONCAT(first_name, ' ', last_name)  
-> FROM citizen  
-> WHERE income/dependents > 10000 AND age > 30;
```

- Full support for SQL GROUP BY and ORDER BY clauses. Support for group functions (COUNT(), COUNT(DISTINCT . . .), AVG(), STD(), SUM(), MAX(), MIN(), and GROUP_CONCAT()).
- Support for LEFT OUTER JOIN and RIGHT OUTER JOIN with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- DELETE, INSERT, REPLACE, and UPDATE return the number of rows that were changed (affected). It is possible to return the number of rows matched instead by setting a flag when connecting to the server.
- The MySQL-specific SHOW statement can be used to retrieve information about databases, storage engines, tables, and indexes.

The EXPLAIN statement can be used to determine how the optimizer resolves a query.

- Function names do not clash with table or column names. For example, ABS is a valid column name. The only restriction is that for a function call, no spaces are allowed between the function name and the ‘(’ that follows it. See [Section 9.5, “Treatment of Reserved Words in MySQL”](#).
- You can mix tables from different databases in the same query (as of MySQL 3.22).

Security:

- A privilege and password system that is very flexible and secure, and that allows host-based verification. Passwords are secure because all password traffic is encrypted when you connect to a server.

Scalability and Limits:

- Handles large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 60,000 tables and about 5,000,000,000 rows.
- Up to 64 indexes per table are allowed (32 before MySQL 4.1.2). Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 1000 bytes (767 for InnoDB); before MySQL 4.1.2, the limit is 500 bytes. An index may use a prefix of a column for CHAR, VARCHAR, BLOB, or TEXT column types.

Connectivity:

- Clients can connect to the MySQL server using TCP/IP sockets on any platform. On Windows systems in the NT family (NT, 2000, XP, 2003, or Vista), clients can connect using named pipes. On Unix systems, clients can connect using Unix domain socket files.
- In MySQL 4.1 and higher, Windows servers also support shared-memory connections if started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=memory` option.
- The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. MyODBC source is available. All ODBC 2.5 functions are supported, as are many others. See [Chapter 23, Connectors](#).
- The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See [Chapter 23, Connectors](#).

- MySQL Connector/NET enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. MySQL Connector/NET is a fully managed ADO.NET driver written in 100% pure C#. See [Chapter 23, Connectors](#).

Localization:

- The server can provide error messages to clients in many languages. See [Section 5.11.2, “Setting the Error Message Language”](#).
- Full support for several different character sets, including latin1 (cp1252), german, big5, ujis, and more. For example, the Scandinavian characters ‘å’, ‘ä’ and ‘ö’ are allowed in table and column names. Unicode support is available as of MySQL 4.1.
- All data is saved in the chosen character set. All comparisons for normal string columns are case-insensitive.
- Sorting is done according to the chosen character set (using Swedish collation by default). It is possible to change this when the MySQL server is started. To see an example of very advanced sorting, look at the Czech sorting code. MySQL Server supports many different character sets that can be specified at compile time and runtime.

Clients and Tools:

- MySQL Server has built-in support for SQL statements to check, optimize, and repair tables. These statements are available from the command line through the **mysqlcheck** client. MySQL also includes **myisamchk**, a very fast command-line utility for performing these operations on MyISAM tables. See [Chapter 5, Database Administration](#).
- All MySQL programs can be invoked with the `--help` or `-?` options to obtain online assistance.

1.4.3. MySQL Stability

This section addresses the questions, “*How stable is MySQL Server?*” and, “*Can*

I depend on MySQL Server in this project?" We will try to clarify these issues and answer some important questions that concern many potential users. The information in this section is based on data gathered from the mailing lists, which are very active in identifying problems as well as reporting types of use.

The original code stems back to the early 1980s. It provides a stable code base, and the ISAM table format used by the original storage engine remains backward-compatible. At TcX, the predecessor of MySQL AB, MySQL code has worked in projects since mid-1996, without any problems. When the MySQL Database Software initially was released to a wider public, our new users quickly found some pieces of untested code. Each new release since then has had fewer portability problems, even though each new release has also had many new features.

Each release of the MySQL Server has been usable. Problems have occurred only when users try code from the "gray zones." Naturally, new users don't know what the gray zones are; this section therefore attempts to document those areas that are currently known. The descriptions mostly deal with Versions 3.23 and later of MySQL Server. All known and reported bugs are fixed in the latest version, with the exception of those listed in the bugs section, which are design-related. See [Section A.8, "Known Issues in MySQL"](#).

The MySQL Server design is multi-layered with independent modules. Some of the newer modules are listed here with an indication of how well-tested each of them is:

- Replication (Stable)

Large groups of servers using replication are in production use, with good results. Work on enhanced replication features is continuing.

- InnoDB tables (Stable)

The InnoDB transactional storage engine has been stable since version 3.23.49. InnoDB is being used in large, heavy-load production systems.

- Full-text searches (Stable)

Full-text searching is widely used. Important feature enhancements were added in MySQL 4.0 and 4.1.

- MyODBC 3.51 (Stable)

MyODBC 3.51 uses ODBC SDK 3.51 and is in wide production use. Some issues brought up appear to be application-related and independent of the ODBC driver or underlying database server.

1.4.4. How Large MySQL Tables Can Be

MySQL 3.22 had a 4GB (4 gigabyte) limit on table size. With the MyISAM storage engine in MySQL 3.23, the maximum table size was increased to 65536 terabytes ($256^7 - 1$ bytes). With this larger allowed table size, the maximum effective table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits.

The InnoDB storage engine maintains InnoDB tables within a tablespace that can be created from several files. This allows a table to exceed the maximum individual file size. The tablespace can include raw disk partitions, which allows extremely large tables. The maximum tablespace size is 64TB.

The following table lists some examples of operating system file-size limits. This is only a rough guide and is not intended to be definitive. For the most up-to-date information, be sure to check the documentation specific to your operating system.

Operating System	File-size Limit
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4+	(using ext3 filesystem) 4TB
Solaris 9/10	16TB
NetWare w/NSS filesystem	8TB
Win32 w/ FAT/FAT32	2GB/4GB
Win32 w/ NTFS	2TB (possibly larger)
MacOS X w/ HFS+	2TB

On Linux 2.2, you can get MyISAM tables larger than 2GB in size by using the Large File Support (LFS) patch for the ext2 filesystem. On Linux 2.4, patches also exist for ReiserFS to get support for big files (up to 2TB). Most current Linux distributions are based on kernel 2.4 or higher and include all the required

LFS patches. With JFS and XFS, petabyte and larger files are possible on Linux. However, the maximum available file size still depends on several factors, one of them being the filesystem used to store MySQL tables.

For a detailed overview about LFS in Linux, have a look at Andreas Jaeger's *Large File Support in Linux* page at http://www.suse.de/~aj/linux_lfs.html.

Windows users please note: FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

By default, MySQL creates MyISAM tables with an internal structure that allows a maximum size of about 4GB. You can check the maximum table size for a MyISAM table with the `SHOW TABLE STATUS` statement or with `myisamchk -dv tbl_name`. See [Section 13.5.4, “SHOW Syntax”](#).

If you need a MyISAM table that is larger than 4GB and your operating system supports large files, the `CREATE TABLE` statement supports `AVG_ROW_LENGTH` and `MAX_ROWS` options. See [Section 13.1.5, “CREATE TABLE Syntax”](#). You can also change these options with `ALTER TABLE` to increase a table's maximum allowable size after the table has been created. See [Section 13.1.2, “ALTER TABLE Syntax”](#).

Other ways to work around file-size limits for MyISAM tables are as follows:

- If your large table is read-only, you can use `myisampack` to compress it. `myisampack` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. `myisampack` also can merge multiple tables into a single table. See [Section 8.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#).
- MySQL includes a MERGE library that allows you to handle a collection of MyISAM tables that have identical structure as a single MERGE table. See [Section 14.3, “The MERGE Storage Engine”](#).

1.4.5. Year 2000 Compliance

The MySQL Server itself has no problems with Year 2000 (Y2K) compliance:

- MySQL Server uses Unix time functions that handle dates into the year 2037 for `TIMESTAMP` values. For `DATE` and `DATETIME` values, dates through the year 9999 are accepted.

- All MySQL date functions are implemented in one source file, `sql/time.cc`, and are coded very carefully to be year 2000-safe.
- In MySQL, the `YEAR` data type can store the years 0 and 1901 to 2155 in one byte and display them using two or four digits. All two-digit years are considered to be in the range 1970 to 2069, which means that if you store 01 in a `YEAR` column, MySQL Server treats it as 2001.

The following simple demonstration illustrates that MySQL Server has no problems with `DATE` or `DATETIME` values through the year 9999, and no problems with `TIMESTAMP` values until after the year 2030:

```
mysql> DROP TABLE IF EXISTS y2k;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE TABLE y2k (date DATE,
->                          date_time DATETIME,
->                          time_stamp TIMESTAMP);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO y2k VALUES
-> ('1998-12-31', '1998-12-31 23:59:59', '1998-12-31 23:59:59'),
-> ('1999-01-01', '1999-01-01 00:00:00', '1999-01-01 00:00:00'),
-> ('1999-09-09', '1999-09-09 23:59:59', '1999-09-09 23:59:59'),
-> ('2000-01-01', '2000-01-01 00:00:00', '2000-01-01 00:00:00'),
-> ('2000-02-28', '2000-02-28 00:00:00', '2000-02-28 00:00:00'),
-> ('2000-02-29', '2000-02-29 00:00:00', '2000-02-29 00:00:00'),
-> ('2000-03-01', '2000-03-01 00:00:00', '2000-03-01 00:00:00'),
-> ('2000-12-31', '2000-12-31 23:59:59', '2000-12-31 23:59:59'),
-> ('2001-01-01', '2001-01-01 00:00:00', '2001-01-01 00:00:00'),
-> ('2004-12-31', '2004-12-31 23:59:59', '2004-12-31 23:59:59'),
-> ('2005-01-01', '2005-01-01 00:00:00', '2005-01-01 00:00:00'),
-> ('2030-01-01', '2030-01-01 00:00:00', '2030-01-01 00:00:00'),
-> ('2040-01-01', '2040-01-01 00:00:00', '2040-01-01 00:00:00'),
-> ('9999-12-31', '9999-12-31 23:59:59', '9999-12-31 23:59:59');
Query OK, 14 rows affected, 2 warnings (0.00 sec)
Records: 14 Duplicates: 0 Warnings: 2
```

```
mysql> SELECT * FROM y2k;
+-----+-----+-----+
| date      | date_time      | time_stamp      |
+-----+-----+-----+
| 1998-12-31 | 1998-12-31 23:59:59 | 1998-12-31 23:59:59 |
| 1999-01-01 | 1999-01-01 00:00:00 | 1999-01-01 00:00:00 |
| 1999-09-09 | 1999-09-09 23:59:59 | 1999-09-09 23:59:59 |
| 2000-01-01 | 2000-01-01 00:00:00 | 2000-01-01 00:00:00 |
| 2000-02-28 | 2000-02-28 00:00:00 | 2000-02-28 00:00:00 |
```

2000-02-29	2000-02-29 00:00:00	2000-02-29 00:00:00
2000-03-01	2000-03-01 00:00:00	2000-03-01 00:00:00
2000-12-31	2000-12-31 23:59:59	2000-12-31 23:59:59
2001-01-01	2001-01-01 00:00:00	2001-01-01 00:00:00
2004-12-31	2004-12-31 23:59:59	2004-12-31 23:59:59
2005-01-01	2005-01-01 00:00:00	2005-01-01 00:00:00
2030-01-01	2030-01-01 00:00:00	2030-01-01 00:00:00
2040-01-01	2040-01-01 00:00:00	0000-00-00 00:00:00
9999-12-31	9999-12-31 23:59:59	0000-00-00 00:00:00

14 rows in set (0.00 sec)

The final two `TIMESTAMP` column values are zero because the year values (2040, 9999) exceed the `TIMESTAMP` maximum. The `TIMESTAMP` data type, which is used to store the current time, supports values that range from '1970-01-01 00:00:00' to '2030-01-01 00:00:00' on 32-bit machines (signed value). On 64-bit machines, `TIMESTAMP` handles values up to 2106 (unsigned value).

Although MySQL Server itself is Y2K-safe, you may run into problems if you use it with applications that are not Y2K-safe. For example, many old applications store or manipulate years using two-digit values (which are ambiguous) rather than four-digit values. This problem may be compounded by applications that use values such as 00 or 99 as “missing” value indicators. Unfortunately, these problems may be difficult to fix because different applications may be written by different programmers, each of whom may use a different set of conventions and date-handling functions.

Thus, even though MySQL Server has no Y2K problems, *it is the application's responsibility to provide unambiguous input*. See [Section 11.3.4, “Y2K Issues and Date Types”](#), for MySQL Server's rules for dealing with ambiguous date input data that contains two-digit year values.

1.5. Overview of the MaxDB Database Management System

MaxDB is a heavy-duty enterprise database. The database management system is SAP-certified.

MaxDB is the new name of a database management system formerly called SAP DB. In 2003 SAP AG and MySQL AB joined a partnership and re-branded the database system to MaxDB. The development of MaxDB has continued since then as it was done before—through the SAP developer team.

MySQL AB cooperates closely with the MaxDB team at SAP around delivering improvements to the MaxDB product. Joint efforts include development of new native drivers to enable more efficient usage of MaxDB in the Open Source community, and improvement of documentation to expand the MaxDB user base. Interoperability features between MySQL and MaxDB database also are seen as important. For example, the new MaxDB Synchronization Manager supports data synchronization from MaxDB to MySQL.

The MaxDB database management system does not share a common code-base with the MySQL database management system. The MaxDB and MySQL database management systems are independent products provided by MySQL AB.

MySQL AB offers a complete portfolio of Professional Services for MaxDB.

1.5.1. What is MaxDB?

MaxDB is an ANSI SQL-92 (entry level) compliant relational database management system (RDBMS) from SAP AG, that is delivered by MySQL AB as well. MaxDB fulfills the needs for enterprise usage: safety, scalability, high concurrency, and performance. It runs on all major operating systems. Over the years it has proven able to run SAP R/3 and terabytes of data in 24×7 operation.

The database development started in 1977 as a research project at the Technical University of Berlin. In the early 1980s it became a database product that subsequently was owned by Nixdorf, Siemens Nixdorf, Software AG, and today

by SAP AG. Along the way, it has been named VDN, Reflex, Supra 2, DDB/4, Entire SQL-DB-Server, and ADABAS D. In 1997, SAP took over the software from Software AG and renamed it to SAP DB. Since October 2000, SAP DB sources additionally were released as Open Source under the GNU General Public License (see [Appendix J, GNU General Public License](#)).

In 2003, SAP AG and MySQL AB formed a partnership and re-branded the database system to MaxDB.

1.5.2. History of MaxDB

The history of MaxDB goes back to SAP DB, SAP AG's DBMS. That is, MaxDB is a re-branded and enhanced version of SAP DB. For many years, MaxDB has been used for small, medium, and large installations of the mySAP Business Suite and other demanding SQL applications requiring an enterprise-class DBMS with regard to the number of users, the transactional workload, and the size of the database.

SAP DB was meant to provide an alternative to third-party database systems such as Oracle, Microsoft SQL Server, and DB2 by IBM. In October 2000, SAP AG released SAP DB under the GNU GPL license (see [Appendix J, GNU General Public License](#)), thus making it Open Source software.

Today, MaxDB is used in about 3,500 SAP customer installations worldwide. Moreover, the majority of all DBMS installations on Unix and Linux within SAP's IT department rely on MaxDB. MaxDB is tuned toward heavy-duty online transaction processing (OLTP) with several thousand users and database sizes ranging from several hundred GB to multiple TB.

In 2003, SAP and MySQL concluded a partnership and development cooperation agreement. As a result, SAP's database system SAP DB has been delivered under the name of MaxDB by MySQL since the release of version 7.5 (November 2003).

Version 7.5 of MaxDB is a direct advancement of the SAP DB 7.4 code base. Therefore, the MaxDB software version 7.5 can be used as a direct upgrade of previous SAP DB versions starting 7.2.04 and higher.

The former SAP DB development team at SAP AG is responsible, now as

before, for developing and supporting MaxDB. MySQL AB cooperates closely with the MaxDB team at SAP around delivering improvements to the MaxDB product, see [Section 1.5, “Overview of the MaxDB Database Management System”](#). Both SAP AG and MySQL AB handle the sale and distribution of MaxDB. The advancement of MaxDB and the MySQL Server leverages synergies that benefit both product lines.

MaxDB is subjected to SAP AG's complete quality assurance process before it is shipped with SAP solutions or provided as a download from the MySQL site.

1.5.3. Features of MaxDB

MaxDB is a heavy-duty, SAP-certified Open Source database for OLTP and OLAP usage which offers high reliability, availability, scalability, and a very comprehensive feature set. It is targeted for large mySAP Business Suite environments and other applications that require maximum enterprise-level database functionality and complements the MySQL database server.

MaxDB operates as a client/server product. It was developed to meet the needs of installations in OLTP and Data Warehouse/OLAP/Decision Support scenarios and offers these benefits:

- **Easy configuration and administration:** GUI-based Installation Manager and Database Manager as single administration tools for DBMS operations
- **Around-the-clock operation, no planned downtimes, no permanent attendance required:** Automatic space management, no need for reorganizations
- **Sophisticated backup and restore capabilities:** Online and incremental backups, recovery wizard to guide you through the recovery scenario
- **Supports large number of users, database sizes in the terabytes, and demanding workloads:** Proven reliability, performance, and scalability
- **High availability:** Cluster support, standby configuration, hot standby configuration

1.5.4. Licensing and Support

MaxDB can be used under the same licenses available for the other products distributed by MySQL AB. Thus, MaxDB is available under the GNU General Public License, and a commercial license. For more information on licensing, see <http://www.mysql.com/company/legal/licensing/>.

MySQL AB offers MaxDB technical support to non-SAP customers. MaxDB support is available on various levels (Basic, Silver, and Gold), which expand from unlimited email/web-support to 24×7 phone support for business critical systems.

MySQL AB also offers Licenses and Support for MaxDB when used with SAP Applications, like SAP NetWeaver and mySAP Business Suite. For more information on licenses and support for your needs, please contact MySQL AB. (See <http://www.mysql.com/company/contact/>.)

Consulting and training services are available. MySQL gives classes on MaxDB at regular intervals. See <http://www.mysql.com/training/> for a list of classes.

1.5.5. Feature Differences Between MaxDB and MySQL

MaxDB is MySQL AB's SAP-certified database. The MaxDB database server complements the MySQL AB product portfolio. Some MaxDB features are not available on the MySQL database management server and vice versa.

The following list summarizes the main differences between MaxDB and MySQL; it is not complete.

- MaxDB runs as a client/server system. MySQL can run as a client/server system or as an embedded system.
- MaxDB might not run on all platforms supported by MySQL.
- MaxDB uses a proprietary network protocol for client/server communication. MySQL uses either TCP/IP (with or without SSL encryption), sockets (under Unix-like systems), or named pipes or shared memory (under Windows NT-family systems).
- MaxDB supports stored procedures and functions. MySQL 5.0 and up also supports stored procedures and functions. MaxDB supports programming of triggers through an SQL extension. MySQL 5.0 supports triggers.

MaxDB contains a debugger for stored procedure languages, can cascade nested triggers, and supports multiple triggers per action and row.

- MaxDB is distributed with user interfaces that are text-based, graphical, or Web-based. MySQL is distributed with text-based user interfaces only; graphical user interfaces such as MySQL Query Browser or MySQL Administrator are shipped separately from the main distributions. Web-based user interfaces for MySQL are offered by third parties.
- MaxDB supports a number of programming interfaces that also are supported by MySQL. For developing with MaxDB, the MaxDB ODBC Driver, SQL Database Connectivity (SQLDBC), JDBC Driver, Perl and Python modules and a MaxDB PHP extension, which provides access to MySQL MaxDB databases using PHP, are available. Third Party Programming Interfaces: Support for OLE DB, ADO, DAO, RDO and .NET through ODBC. MaxDB supports embedded SQL with C/C++.
- MaxDB includes administrative features that MySQL does not have: job scheduling by time (included in MySQL as of 5.1), event, and alert, and sending messages to a database administrator on alert thresholds. (MySQL has scheduling support starting with version 5.1.6.)

1.5.6. Interoperability Features Between MaxDB and MySQL

MaxDB and MySQL are independent database management servers. The interoperation of the systems is possible in a way that the systems can exchange their data. To exchange data between MaxDB and MySQL, you can use the import and export tools of the systems or the MaxDB Synchronization Manager. The import and export tools can be used to transfer data in an infrequent, manual fashion. The MaxDB Synchronization Manager offers faster, automatic data transfer capabilities.

The MaxDB Loader can be used to export data and object definitions. The Loader can export data using MaxDB internal, binary formats and text formats (CSV). Data exported from MaxDB in text formats can be imported into MySQL using the **mysqlimport** client program. To export MySQL data, you can use either **mysqldump** to create INSERT statements or `SELECT . . . INTO OUTFILE` to create a text file (CSV). Use the MaxDB Loader to import the data files generated by MySQL.

Object definitions can be exchanged between the systems using MaxDB Loader and the MySQL tool **mysqldump**. As the SQL dialects of both systems differ slightly and MaxDB has features currently not supported by MySQL like SQL constraints, we recommend to hand-tune the definition files. The **mysqldump** tool offers an option `--compatible=maxdb` to produce output that is compatible with MaxDB to make porting easier.

The MaxDB Synchronization Manager is available as part of MaxDB 7.6. The Synchronization Manager supports creation of asynchronous replication scenarios between several MaxDB instances. However, interoperability features also are planned, so that the Synchronization Manager supports replication to and from a MySQL server.

1.5.7. MaxDB-Related Links

The main page for MaxDB information is <http://www.mysql.com/products/maxdb>, which provides details about the features of the MaxDB database management systems and has pointers to available documentation.

The MySQL Reference Manual does not contain any MaxDB documentation other than the introduction given in this section. MaxDB has its own documentation, which is called the MaxDB library and is available at <http://dev.mysql.com/doc/maxdb/index.html>.

MySQL AB runs a community mailing list on MaxDB; see <http://lists.mysql.com/maxdb>. The list shows a vivid community discussion. Many of the core developers contribute to it. Product announcements are sent to the list.

A Web forum on MaxDB is available at <http://forums.mysql.com/>. The forum focuses on MaxDB questions not related to SAP applications.

1.6. MySQL Development Roadmap

This section provides a snapshot of the MySQL development roadmap, including major features implemented in or planned for various MySQL releases. The following sections provide information for each release series.

The current production release series is MySQL 5.0, which was declared stable for production use as of MySQL 5.0.15, released in October 2005. The previous production release series was MySQL 4.1, which was declared stable for production use as of MySQL 4.1.7, released in October 2004. “Production status” means that future 5.0 and 4.1 development is limited only to bugfixes. For the older MySQL 4.0 and 3.23 series, only critical bugfixes are made.

Active MySQL development is currently taking place in the MySQL 5.0 and 5.1 release series, and new features are being added only to the latter.

Before upgrading from one release series to the next, please see the notes in [Section 2.11, “Upgrading MySQL”](#).

The most requested features and the versions in which they were implemented or are scheduled for implementation are summarized in the following table:

Feature	MySQL Series
Foreign keys	3.23 (for the InnoDB storage engine)
Unions	4.0
Subqueries	4.1
R-trees	4.1 (for the MyISAM storage engine)
Stored procedures	5.0
Views	5.0
Cursors	5.0
XA transactions	5.0
Foreign keys	5.2 (implemented in 3.23 for InnoDB)
Triggers	5.0 and 5.1
Partitioning	5.1

Pluggable Storage Engine API	5.1
Row-Based Replication	5.1

1.6.1. What's New in MySQL 5.0

The following features are implemented in MySQL 5.0.

- **BIT Data Type:** Can be used to store numbers in binary notation. See [Section 11.1.1, “Overview of Numeric Types”](#).
- **Cursors:** Elementary support for server-side cursors. For information about using cursors within stored routines, see [Section 17.2.9, “Cursors”](#). For information about using cursors from within the C API, see [Section 22.2.7.3, “mysql_stmt_attr_set\(\)”](#).
- **Information Schema:** The introduction of the INFORMATION_SCHEMA database in MySQL 5.0 provided a standards-compliant means for accessing the MySQL Server's metadata; that is, data about the databases (schemas) on the server and the objects which they contain. See [Chapter 20, The INFORMATION_SCHEMA Database](#).
- **Instance Manager:** Can be used to start and stop the MySQL Server, even from a remote host. See [Section 5.5, “mysqlmanager — The MySQL Instance Manager”](#).
- **Precision Math:** MySQL 5.0 introduced stricter criteria for acceptance or rejection of data, and implemented a new library for fixed-point arithmetic. These contributed to a much higher degree of accuracy for mathematical operations and greater control over invalid values. See [Chapter 21, Precision Math](#).
- **Storage Engines:** Storage engines added in MySQL 5.0 include ARCHIVE and FEDERATED. See [Section 14.8, “The ARCHIVE Storage Engine”](#), and [Section 14.7, “The FEDERATED Storage Engine”](#).
- **Stored Routines:** Support for named stored procedures and stored functions was implemented in MySQL 5.0. See [Chapter 17, Stored Procedures and Functions](#).

- **Strict Mode and Standard Error Handling:** MySQL 5.0 added a strict mode where by it follows standard SQL in a number of ways in which it did not previously. Support for standard SQLSTATE error messages was also implemented. See [Section 5.2.5, “The Server SQL Mode”](#).
- **Triggers:** MySQL 5.0 added limited support for triggers. See [Chapter 18, Triggers](#), and [Section 1.9.5.4, “Stored Routines and Triggers”](#).
- **VARCHAR Data Type:** The maximum effective length of a VARCHAR column was increased to 65,532 bytes, and stripping of trailing whitespace was eliminated. See [Section 11.4, “String Types”](#).
- **Views:** MySQL 5.0 added support for named, updatable views. See [Chapter 19, Views](#), and [Section 1.9.5.6, “Views”](#).
- **XA Transactions:** See [Section 13.4.7, “XA Transactions”](#).
- **Performance enhancements:** A number of improvements were made in MySQL 5.0 to improve the speed of certain types of queries and in the handling of certain types. These include:
 - MySQL 5.0 introduces a new “greedy” optimizer which can greatly reduce the time required to arrive at a query execution plan. This is particularly noticeable where several tables are to be joined and no good join keys can otherwise be found. Without the greedy optimizer, the complexity of the search for an execution plan is calculated as $N!$, where N is the number of tables to be joined. The greedy optimizer reduces this to $N!/(D-1)!$, where D is the depth of the search. Although the greedy optimizer does not guarantee the best possible of all execution plans (this is currently being worked on), it can reduce the time spent arriving at an execution plan for a join involving a great many tables — 30, 40, or more — by a factor of as much as 1,000. This should eliminate most if not all situations where users thought that the optimizer had hung when trying to perform joins across many tables.
 - Use of the *Index Merge* method to obtain better optimization of AND and OR relations over different keys. (Previously, these were optimized only where both relations in the WHERE clause involved the same key.) This also applies to other one-to-one comparison operators (>, <, and

so on), including = and the IN operator. This means that MySQL can use multiple indexes in retrieving results for conditions such as WHERE key1 > 4 OR key2 < 7 and even combinations of conditions such as WHERE (key1 > 4 OR key2 < 7) AND (key3 >= 10 OR key4 = 1). See [Section 7.2.6, “Index Merge Optimization”](#).

- A new equality detector finds and optimizes “hidden” equalities in joins. For example, a WHERE clause such as

```
t1.c1=t2.c2 AND t2.c2=t3.c3 AND t1.c1 < 5
```

implies these other conditions

```
t1.c1=t3.c3 AND t2.c2 < 5 AND t3.c3 < 5
```

These optimizations can be applied with any combination of AND and OR operators. See [Section 7.2.10, “Nested Join Optimization”](#), and [Section 7.2.11, “Outer Join Simplification”](#).

- Optimization of NOT IN and NOT BETWEEN relations, reducing or eliminating table scans for queries making use of them by means of range analysis. The performance of MySQL with regard to these relations now matches its performance with regard to IN and BETWEEN.
- The VARCHAR data type as implemented in MySQL 5.0 is more efficient than in previous versions, due to the elimination of the old (and nonstandard) removal of trailing spaces during retrieval.
- The addition of a true BIT column type; this type is much more efficient for storage and retrieval of Boolean values than the workarounds required in MySQL in versions previous to 5.0.
- **Performance Improvements in the InnoDB Storage Engine:**
 - New compact storage format which can save up to 20% of the disk space required in previous MySQL/InnoDB versions.
 - Faster recovery from a failed or aborted ALTER TABLE.
 - Faster implementation of TRUNCATE.

(See [Section 14.2, “The InnoDB Storage Engine”](#).)

- **Performance Improvements in the NDBCluster Storage Engine:**

- Faster handling of queries that use IN and BETWEEN.
- **Condition pushdown:** In cases involving the comparison of an unindexed column with a constant, this condition is “pushed down” to the cluster where it is evaluated in all partitions simultaneously, eliminating the need to send non-matching records over the network. This can make such queries 10 to 100 times faster than in MySQL 4.1 Cluster.

See [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#), for more information.

(See [Chapter 15, MySQL Cluster](#).)

For those wishing to take a look at the bleeding edge of MySQL development, we make our BitKeeper repository for MySQL publicly available. See [Section 2.9.3, “Installing from the Development Source Tree”](#).

1.7. MySQL Information Sources

This section lists sources of additional information that you may find helpful, such as the MySQL mailing lists and user forums, and Internet Relay Chat.

1.7.1. MySQL Mailing Lists

This section introduces the MySQL mailing lists and provides guidelines as to how the lists should be used. When you subscribe to a mailing list, you receive all postings to the list as email messages. You can also send your own questions and answers to the list.

To subscribe to or unsubscribe from any of the mailing lists described in this section, visit <http://lists.mysql.com/>. For most of them, you can select the regular version of the list where you get individual messages, or a digest version where you get one large message per day.

Please *do not* send messages about subscribing or unsubscribing to any of the mailing lists, because such messages are distributed automatically to thousands of other users.

Your local site may have many subscribers to a MySQL mailing list. If so, the site may have a local mailing list, so that messages sent from `lists.mysql.com` to your site are propagated to the local list. In such cases, please contact your system administrator to be added to or dropped from the local MySQL list.

If you wish to have traffic for a mailing list go to a separate mailbox in your mail program, set up a filter based on the message headers. You can use either the `List-ID:` or `Delivered-To:` headers to identify list messages.

The MySQL mailing lists are as follows:

- announce

This list is for announcements of new versions of MySQL and related programs. This is a low-volume list to which all MySQL users should subscribe.

- `mysql`

This is the main list for general MySQL discussion. Please note that some topics are better discussed on the more-specialized lists. If you post to the wrong list, you may not get an answer.

- `bugs`

This list is for people who want to stay informed about issues reported since the last release of MySQL or who want to be actively involved in the process of bug hunting and fixing. See [Section 1.8, “How to Report Bugs or Problems”](#).

- `internals`

This list is for people who work on the MySQL code. This is also the forum for discussions on MySQL development and for posting patches.

- `mysqldoc`

This list is for people who work on the MySQL documentation: people from MySQL AB, translators, and other community members.

- `benchmarks`

This list is for anyone interested in performance issues. Discussions concentrate on database performance (not limited to MySQL), but also include broader categories such as performance of the kernel, filesystem, disk system, and so on.

- `packagers`

This list is for discussions on packaging and distributing MySQL. This is the forum used by distribution maintainers to exchange ideas on packaging MySQL and on ensuring that MySQL looks and feels as similar as possible on all supported platforms and operating systems.

- `java`

This list is for discussions about the MySQL server and Java. It is mostly

used to discuss JDBC drivers such as MySQL Connector/J.

- win32

This list is for all topics concerning the MySQL software on Microsoft operating systems, such as Windows 9x, Me, NT, 2000, XP, and 2003.

- myodbc

This list is for all topics concerning connecting to the MySQL server with ODBC.

- gui-tools

This list is for all topics concerning MySQL graphical user interface tools such as MySQL Administrator and MySQL Query Browser.

- cluster

This list is for discussion of MySQL Cluster.

- dotnet

This list is for discussion of the MySQL server and the .NET platform. It is mostly related to MySQL Connector/Net.

- plusplus

This list is for all topics concerning programming with the C++ API for MySQL.

- perl

This list is for all topics concerning Perl support for MySQL with DBD::mysql.

If you're unable to get an answer to your questions from a MySQL mailing list or forum, one option is to purchase support from MySQL AB. This puts you in direct contact with MySQL developers.

The following table shows some MySQL mailing lists in languages other than

English. These lists are not operated by MySQL AB.

- [<mysql-france-subscribe@yahoogroups.com>](mailto:mysql-france-subscribe@yahoogroups.com)

A French mailing list.

- [<list@tinc.net>](mailto:list@tinc.net)

A Korean mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

- [<mysql-de-request@lists.4t2.com>](mailto:mysql-de-request@lists.4t2.com)

A German mailing list. To subscribe, email `subscribe mysql-de your@email.address` to this list. You can find information about this mailing list at <http://www.4t2.com/mysql/>.

- [<mysql-br-request@listas.linkway.com.br>](mailto:mysql-br-request@listas.linkway.com.br)

A Portuguese mailing list. To subscribe, email `subscribe mysql-br your@email.address` to this list.

- [<mysql-alta@elistas.net>](mailto:mysql-alta@elistas.net)

A Spanish mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

1.7.1.1. Guidelines for Using the Mailing Lists

Please don't post mail messages from your browser with HTML mode turned on. Many users don't read mail with a browser.

When you answer a question sent to a mailing list, if you consider your answer to have broad interest, you may want to post it to the list instead of replying directly to the individual who asked. Try to make your answer general enough that people other than the original poster may benefit from it. When you post to the list, please make sure that your answer is not a duplication of a previous answer.

Try to summarize the essential part of the question in your reply. Don't feel

obliged to quote the entire original message.

When answers are sent to you individually and not to the mailing list, it is considered good etiquette to summarize the answers and send the summary to the mailing list so that others may have the benefit of responses you received that helped you solve your problem.

1.7.2. MySQL Community Support at the MySQL Forums

The forums at <http://forums.mysql.com> are an important community resource. Many forums are available, grouped into these general categories:

- Migration
- MySQL Usage
- MySQL Connectors
- Programming Languages
- Tools
- 3rd-Party Applications
- Storage Engines
- MySQL Technology
- SQL Standards
- Business

1.7.3. MySQL Community Support on Internet Relay Chat (IRC)

In addition to the various MySQL mailing lists and forums, you can find experienced community people on Internet Relay Chat (IRC). These are the best networks/channels currently known to us:

freenode (see <http://www.freenode.net/> for servers)

- #mysql is primarily for MySQL questions, but other database and general SQL questions are welcome. Questions about PHP, Perl, or C in combination with MySQL are also common.

If you are looking for IRC client software to connect to an IRC network, take a look at xChat (<http://www.xchat.org/>). X-Chat (GPL licensed) is available for Unix as well as for Windows platforms (a free Windows build of X-Chat is available at <http://www.silverex.org/download/>).

1.8. How to Report Bugs or Problems

Before posting a bug report about a problem, please try to verify that it is a bug and that it has not been reported already:

- Start by searching the MySQL online manual at <http://dev.mysql.com/doc/>. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. The change history (<http://dev.mysql.com/doc/mysql/en/news.html>) can be particularly useful since it is quite possible that a newer version contains a solution to your problem.
- If you get a parse error for a SQL statement, please check your syntax closely. If you can't find something wrong with it, it's extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual doesn't cover the syntax that you are using, MySQL Server doesn't support your statement. In this case, your options are to implement the syntax yourself or email <licensing@mysql.com> and ask for an offer to implement it.

If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server.

- For solutions to some common problems, see [Appendix A, Problems and Common Errors](#).
- Search the bugs database at <http://bugs.mysql.com/> to see whether the bug has been reported and fixed.
- Search the MySQL mailing list archives at <http://lists.mysql.com/>. See [Section 1.7.1, “MySQL Mailing Lists”](#).
- You can also use <http://www.mysql.com/search/> to search all the Web pages (including the manual) that are located at the MySQL AB Web site.

If you can't find an answer in the manual, the bugs database, or the mailing list

archives, check with your local MySQL expert. If you still can't find an answer to your question, please use the following guidelines for reporting the bug.

The normal way to report bugs is to visit <http://bugs.mysql.com/>, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports. If you have no Web access, you can generate a bug report by using the **mysqlbug** script described at the end of this section.

Bugs posted in the bugs database at <http://bugs.mysql.com/> that are corrected for a given release are noted in the change history.

If you have found a sensitive security bug in MySQL, you can send email to [<security@mysql.com>](mailto:security@mysql.com).

To discuss problems with other users, you can use one of the MySQL mailing lists. [Section 1.7.1, “MySQL Mailing Lists”](#).

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release. This section helps you write your report correctly so that you don't waste your time doing things that may not help us much or at all. Please read this section carefully and make sure that all the information described here is included in your report.

Preferably, you should test the problem using the latest production or development version of MySQL Server before posting. Anyone should be able to repeat the bug by just using `mysql test < script_file` on your test case or by running the shell or Perl script that you include in the bug report. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself. The best reports are those that include a full example showing how to reproduce the bug or problem. See [Section E.1.6, “Making a Test Case If You Experience Table Corruption”](#).

Remember that it is possible for us to respond to a report containing too much

information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details don't matter. A good principle to follow is that if you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution that you use, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). These are highly relevant pieces of information, and in 99 cases out of 100, the bug report is useless without them. Very often we get questions like, "Why doesn't this work for me?" Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Errors often are platform-dependent. In such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If you compiled MySQL from source, remember also to provide information about your compiler if it is related to the problem. Often people find bugs in compilers and think the problem is MySQL-related. Most compilers are under development all the time and become better version by version. To determine whether your problem depends on your compiler, we need to know what compiler you used. Note that every compiling problem should be regarded as a bug and reported accordingly.

If a program produces an error message, it is very important to include the message in your report. If we try to search for something from the archives, it is better that the error message reported exactly matches the one that the program produces. (Even the lettercase should be observed.) It is best to copy and paste the entire error message into your report. You should never try to reproduce the message from memory.

If you have a problem with Connector/ODBC (MyODBC), please try to generate a trace file and send it with your report. See the MyODBC section of [Chapter 23, Connectors](#).

If your report includes long query output lines from test cases that you run with the **mysql** command-line tool, you can make the output more readable by using

the `--vertical` option or the `\G` statement terminator. The `EXPLAIN SELECT` example later in this section demonstrates the use of `\G`.

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 5.0.19). You can find out which version you are running by executing **mysqladmin version**. The **mysqladmin** program can be found in the `bin` directory under your MySQL installation directory.
- The manufacturer and model of the machine on which you experience the problem.
- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the “Help/About Windows” menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.
- If you are using a source distribution of the MySQL software, include the name and version number of the compiler that you used. If you have a binary distribution, include the distribution name.
- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.
- If **mysqld** died, you should also report the statement that crashed **mysqld**. You can usually get this information by running **mysqld** with query logging enabled, and then looking in the log after **mysqld** crashes. See [Section E.1.5, “Using Server Logs to Find Causes of Errors in **mysqld**”](#).
- If a database table is related to the problem, include the output from the `SHOW CREATE TABLE db_name.tbl_name` statement in the bug report. This is a very easy way to get the definition of any table in a database. The information helps us create a situation matching the one that you have experienced.

- For performance-related bugs or problems with `SELECT` statements, you should always include the output of `EXPLAIN SELECT ...`, and at least the number of rows that the `SELECT` statement produces. You should also include the output from `SHOW CREATE TABLE tbl_name` for each table that is involved. The more information you provide about your situation, the more likely it is that someone can help you.

The following is an example of a very good bug report. The statements are run using the **mysql** command-line tool. Note the use of the `\G` statement terminator for statements that would otherwise provide very long output lines that are difficult to read.

```
mysql> SHOW VARIABLES;
mysql> SHOW COLUMNS FROM ... \G
      <output from SHOW COLUMNS>
mysql> EXPLAIN SELECT ... \G
      <output from EXPLAIN>
mysql> FLUSH STATUS;
mysql> SELECT ...;
      <A short version of the output from SELECT,
      including the time taken to run the query>
mysql> SHOW STATUS;
      <output from SHOW STATUS>
```

- If a bug or problem occurs while running **mysqld**, try to provide an input script that reproduces the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better. If you can make a reproducible test case, you should upload it to be attached to the bug report.

If you can't provide a script, you should at least include the output from **mysqladmin variables extended-status processlist** in your report to provide some information on how your system is performing.

- If you can't produce a test case with only a few rows, or if the test table is too big to be included in the bug report (more than 10 rows), you should dump your tables using **mysqldump** and create a README file that describes your problem. Create a compressed archive of your files using **tar** and **gzip** or **zip**, and use FTP to transfer the archive to <ftp://ftp.mysql.com/pub/mysql/upload/>. Then enter the problem into our bugs database at <http://bugs.mysql.com/>.

- If you believe that the MySQL server produces a strange result from a statement, include not only the result, but also your opinion of what the result should be, and an explanation describing the basis for your opinion.
- When you provide an example of the problem, it's better to use the table names, variable names, and so forth that exist in your actual situation than to come up with new names. The problem could be related to the name of a table or variable. These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation, and it is by all means better for us. If you have data that you don't want to be visible to others in the bug report, you can use FTP to transfer it to <ftp://ftp.mysql.com/pub/mysql/upload/>. If the information is really top secret and you don't want to show it even to us, go ahead and provide an example using other names, but please regard this as the last choice.
- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the **mysqld** server, as well as the options that you use to run any MySQL client programs. The options to programs such as **mysqld** and **mysql**, and to the **configure** script, are often key to resolving problems and are very relevant. It is never a bad idea to include them. If your problem involves a program written in a language such as Perl or PHP, please include the language processor's version number, as well as the version for any modules that the program uses. For example, if you have a Perl script that uses the DBI and DBD::mysql modules, include the version numbers for Perl, DBI, and DBD::mysql.
- If your question is related to the privilege system, please include the output of **mysqlaccess**, the output of **mysqladmin reload**, and all the error messages you get when trying to connect. When you test your privileges, you should first run **mysqlaccess**. After this, execute **mysqladmin reload version** and try to connect with the program that gives you trouble. **mysqlaccess** can be found in the bin directory under your MySQL installation directory.
- If you have a patch for a bug, do include it. But don't assume that the patch is all we need, or that we can use it, if you don't provide some necessary information such as test cases showing the bug that your patch fixes. We

might find problems with your patch or we might not understand it at all. If so, we can't use it.

If we can't verify the exact purpose of the patch, we won't use it. Test cases help us here. Show that the patch handles all the situations that may occur. If we find a borderline case (even a rare one) where the patch won't work, it may be useless.

- Guesses about what the bug is, why it occurs, or what it depends on are usually wrong. Even the MySQL team can't guess such things without first using a debugger to determine the real cause of a bug.
- Indicate in your bug report that you have checked the reference manual and mail archive so that others know you have tried to solve the problem yourself.
- If the problem is that your data appears corrupt or you get errors when you access a particular table, you should first check your tables and then try to repair them with `CHECK TABLE` and `REPAIR TABLE` or with **myisamchk**. See [Chapter 5, Database Administration](#).

If you are running Windows, please verify the value of `lower_case_table_names` using the `SHOW VARIABLES LIKE 'lower_case_table_names'` command. This variable affects how the server handles lettercase of database and table names. Its effect for a given value should be as described in [Section 9.2.2, “Identifier Case Sensitivity”](#).

- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the `.err` suffix in the name.) See [Section 5.12.1, “The Error Log”](#). Please include any relevant information from this file in your bug report. Normally **mysqld** should *never* crash a table if nothing killed it in the middle of an update. If you can find the cause of **mysqld** dying, it's much easier for us to provide you with a fix for the problem. See [Section A.1, “How to Determine What Is Causing a Problem”](#).
- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software are thoroughly tested and should work without problems. We

believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See [Section 2.1.2, “Choosing Which MySQL Distribution to Install”](#).

If you have no Web access and cannot report a bug by visiting <http://bugs.mysql.com/>, you can use the **mysqlbug** script to generate a bug report (or a report about any problem). **mysqlbug** helps you generate a report by determining much of the following information automatically, but if something important is missing, please include it with your message. **mysqlbug** can be found in the `scripts` directory (source distribution) and in the `bin` directory under your MySQL installation directory (binary distribution).

1.9. MySQL Standards Compliance

This section describes how MySQL relates to the ANSI/ISO SQL standards. MySQL Server has many extensions to the SQL standard, and here you can find out what they are and how to use them. You can also find information about functionality missing from MySQL Server, and how to work around some of the differences.

The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” or “standard SQL” to mean the current version of the SQL Standard at any time.

One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability. We are not afraid to add extensions to SQL or support for non-SQL features if this greatly increases the usability of MySQL Server for a large segment of our user base. The HANDLER interface is an example of this strategy. See [Section 13.2.3, “HANDLER Syntax”](#).

We continue to support transactional and non-transactional databases to satisfy both mission-critical 24/7 usage and heavy Web or logging usage.

MySQL Server was originally designed to work with medium-sized databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-sized databases, but the code can also be compiled in a reduced version suitable for hand-held and embedded devices. The compact design of the MySQL server makes development in both directions possible without any conflicts in the source tree.

Currently, we are not targeting real-time support, although MySQL replication capabilities offer significant functionality.

MySQL supports high-availability database clustering using the NDBCluster storage engine. See [Chapter 15, MySQL Cluster](#).

XML support is to be implemented in a future version of the database server.

1.9.1. What Standards MySQL Follows

Our aim is to support the full ANSI/ISO SQL standard, but without making concessions to speed and quality of the code.

ODBC levels 0-3.51.

1.9.2. Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differentially for different clients. This capability enables each application to tailor the server's operating mode to its own requirements.

SQL modes control aspects of server operation such as what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting **mysqld** with the `--sql-mode="mode_value"` option. Beginning with MySQL 4.1, you can also change the mode at runtime by setting the `sql_mode` system variable with a `SET [SESSION|GLOBAL] sql_mode='mode_value'` statement.

For more information on setting the SQL mode, see [Section 5.2.5, “The Server SQL Mode”](#).

1.9.3. Running MySQL in ANSI Mode

You can tell **mysqld** to run in ANSI mode with the `--ansi` startup option. Running the server in ANSI mode is the same as starting it with the following options:

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

As of MySQL 4.1.1, you can achieve the same effect at runtime by executing these two statements:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SET GLOBAL sql_mode = 'ANSI';
```

You can see that setting the `sql_mode` system variable to 'ANSI' enables all SQL mode options that are relevant for ANSI mode as follows:

```
mysql> SET GLOBAL sql_mode='ANSI';
mysql> SELECT @@global.sql_mode;
-> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,A
```

Note that running the server in ANSI mode with `--ansi` is not quite the same as setting the SQL mode to 'ANSI'. The `--ansi` option affects the SQL mode and also sets the transaction isolation level. Setting the SQL mode to 'ANSI' has no effect on the isolation level.

See [Section 5.2.1, “mysqld Command Options”](#), and [Section 1.9.2, “Selecting SQL Modes”](#).

1.9.4. MySQL Extensions to Standard SQL

MySQL Server supports some extensions that you probably won't find in other SQL DBMSs. Be warned that if you use them, your code won't be portable to other SQL servers. In some cases, you can write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the '!' character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

MySQL Server maps each database to a directory under the MySQL data directory, and maps tables within a database to filenames in the database directory. This has a few implications:

- Database and table names are case sensitive in MySQL Server on operating systems that have case-sensitive filenames (such as most Unix systems). See [Section 9.2.2, “Identifier Case Sensitivity”](#).
- You can use standard system commands to back up, rename, move, delete, and copy tables that are managed by the MyISAM storage engine. For example, it is possible to rename a MyISAM table by renaming the .MYD, .MYI, and .frm files to which the table corresponds. (Nevertheless, it is preferable to use `RENAME TABLE` or `ALTER TABLE ... RENAME` and let the server rename the files.)

Database and table names cannot contain pathname separator characters ('/', '\').

- General language syntax
 - By default, strings can be enclosed by either “” or ‘ ’, not just by ‘ ’. (If the `ANSI_QUOTES` SQL mode is enabled, strings can be enclosed only by ‘ ’ and the server interprets strings enclosed by “” as identifiers.)
 - ‘\’ is the escape character in strings.
 - In SQL statements, you can access tables from different databases with the `db_name.tbl_name` syntax. Some SQL servers provide the same functionality but call this `user space`. MySQL Server doesn't support tablespaces such as used in statements like this: `CREATE TABLE ralph.my_table ... IN my_tablespace`.
- SQL statement syntax
 - The `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
 - The `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE` statements. See [Section 13.1.3, “CREATE DATABASE Syntax”](#),

[Section 13.1.6, “DROP DATABASE Syntax”](#), and [Section 13.1.1, “ALTER DATABASE Syntax”](#).

- The DO statement.
- EXPLAIN SELECT to obtain a description of how tables are processed by the query optimizer.
- The FLUSH and RESET statements.
- The SET statement. See [Section 13.5.3, “SET Syntax”](#).
- The SHOW statement. See [Section 13.5.4, “SHOW Syntax”](#). As of MySQL 5.0, the information produced by many of the MySQL-specific SHOW statements can be obtained in more standard fashion by using SELECT to query INFORMATION_SCHEMA. See [Chapter 20, *The INFORMATION_SCHEMA Database*](#).
- Use of LOAD DATA INFILE. In many cases, this syntax is compatible with Oracle's LOAD DATA INFILE. See [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).
- Use of RENAME TABLE. See [Section 13.1.9, “RENAME TABLE Syntax”](#).
- Use of REPLACE instead of DELETE plus INSERT. See [Section 13.2.6, “REPLACE Syntax”](#).
- Use of CHANGE col_name, DROP col_name, or DROP INDEX, IGNORE or RENAME in ALTER TABLE statements. Use of multiple ADD, ALTER, DROP, or CHANGE clauses in an ALTER TABLE statement. See [Section 13.1.2, “ALTER TABLE Syntax”](#).
- Use of index names, indexes on a prefix of a column, and use of INDEX or KEY in CREATE TABLE statements. See [Section 13.1.5, “CREATE TABLE Syntax”](#).
- Use of TEMPORARY or IF NOT EXISTS with CREATE TABLE.
- Use of IF EXISTS with DROP TABLE and DROP DATABASE.

- The capability of dropping multiple tables with a single `DROP TABLE` statement.
- The `ORDER BY` and `LIMIT` clauses of the `UPDATE` and `DELETE` statements.
- `INSERT INTO tbl_name SET col_name = ...` syntax.
- The `DELAYED` clause of the `INSERT` and `REPLACE` statements.
- The `LOW_PRIORITY` clause of the `INSERT`, `REPLACE`, `DELETE`, and `UPDATE` statements.
- Use of `INTO OUTFILE` or `INTO DUMPFILE` in `SELECT` statements. See [Section 13.2.7, “SELECT Syntax”](#).
- Options such as `STRAIGHT_JOIN` or `SQL_SMALL_RESULT` in `SELECT` statements.
- You don't need to name all selected columns in the `GROUP BY` clause. This gives better performance for some very specific, but quite normal queries. See [Section 12.10, “Functions and Modifiers for Use with GROUP BY Clauses”](#).
- You can specify `ASC` and `DESC` with `GROUP BY`, not just with `ORDER BY`.
- The ability to set variables in a statement with the `:=` assignment operator:

```
mysql> SELECT @a:=SUM(total),@b=COUNT(*),@a/@b AS avg
-> FROM test_table;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

- Data types
 - The `MEDIUMINT`, `SET`, and `ENUM` data types, and the various `BLOB` and `TEXT` data types.
 - The `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED`, and `ZEROFILL` data type attributes.
- Functions and operators

- To make it easier for users who migrate from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.
- MySQL Server understands the `||` and `&&` operators to mean logical OR and AND, as in the C programming language. In MySQL Server, `||` and OR are synonyms, as are `&&` and AND. Because of this nice syntax, MySQL Server doesn't support the standard SQL `||` operator for string concatenation; use `CONCAT()` instead. Because `CONCAT()` takes any number of arguments, it's easy to convert use of the `||` operator to MySQL Server.
- Use of `COUNT(DISTINCT value_list)` where *value_list* has more than one element.
- String comparisons are case-insensitive by default, with sort ordering determined by collation of the current character set, which is `latin1 (cp1252 West European)` by default. If you don't like this, you should declare your columns with the `BINARY` attribute or use the `BINARY` cast, which causes comparisons to be done using the underlying character code values rather than a lexical ordering.
- The `%` operator is a synonym for `MOD()`. That is, `N % M` is equivalent to `MOD(N,M)`. `%` is supported for C programmers and for compatibility with PostgreSQL.
- The `=`, `<>`, `<=>`, `<`, `>`, `<<`, `>>`, `<=>`, AND, OR, or LIKE operators may be used in expressions in the output column list (to the left of the FROM) in SELECT statements. For example:


```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```
- The `LAST_INSERT_ID()` function returns the most recent `AUTO_INCREMENT` value. See [Section 12.9.3, “Information Functions”](#).
- LIKE is allowed on numeric values.
- The `REGEXP` and `NOT REGEXP` extended regular expression operators.
- `CONCAT()` or `CHAR()` with one argument or more than two arguments.

(In MySQL Server, these functions can take a variable number of arguments.)

- The BIT_COUNT(), CASE, ELT(), FROM_DAYS(), FORMAT(), IF(), PASSWORD(), ENCRYPT(), MD5(), ENCODE(), DECODE(), PERIOD_ADD(), PERIOD_DIFF(), TO_DAYS(), and WEEKDAY() functions.
- Use of TRIM() to trim substrings. Standard SQL supports removal of single characters only.
- The GROUP BY functions STD(), BIT_OR(), BIT_AND(), BIT_XOR(), and GROUP_CONCAT(). See [Section 12.10, “Functions and Modifiers for Use with GROUP BY Clauses”](#).

For a prioritized list indicating when new extensions are added to MySQL Server, you should consult the online MySQL development roadmap at <http://dev.mysql.com/doc/mysql/en/roadmap.html>.

1.9.5. MySQL Differences from Standard SQL

We try to make MySQL Server follow the ANSI SQL standard and the ODBC SQL standard, but MySQL Server performs operations differently in some cases:

- For VARCHAR columns, trailing spaces are removed when the value is stored. (This is fixed in MySQL 5.0.3). See [Section A.8, “Known Issues in MySQL”](#).
- In some cases, CHAR columns are silently converted to VARCHAR columns when you define a table or alter its structure. (This is fixed in MySQL 5.0.3). See [Section 13.1.5.1, “Silent Column Specification Changes”](#).
- There are several differences between the MySQL and standard SQL privilege systems. For example, in MySQL, privileges for a table are not automatically revoked when you delete a table. You must explicitly issue a REVOKE statement to revoke privileges for a table. For more information, see [Section 13.5.1.5, “REVOKE Syntax”](#).
- The CAST() function does not support cast to REAL or BIGINT. See [Section 12.8, “Cast Functions and Operators”](#).

- Standard SQL requires that a HAVING clause in a SELECT statement be able to refer to columns in the GROUP BY clause. This cannot be done before MySQL 5.0.2.

1.9.5.1. Subquery Support

MySQL 4.1 and up supports subqueries and derived tables. A “subquery” is a SELECT statement nested within another statement. A “derived table” (an unnamed view) is a subquery in the FROM clause of another statement. See [Section 13.2.8, “Subquery Syntax”](#).

For MySQL versions older than 4.1, most subqueries can be rewritten using joins or other methods. See [Section 13.2.8.11, “Rewriting Subqueries as Joins for Earlier MySQL Versions”](#), for examples that show how to do this.

1.9.5.2. SELECT INTO TABLE

MySQL Server doesn't support the SELECT ... INTO TABLE Sybase SQL extension. Instead, MySQL Server supports the INSERT INTO ... SELECT standard SQL syntax, which is basically the same thing. See [Section 13.2.4.1, “INSERT ... SELECT Syntax”](#). For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Alternatively, you can use SELECT ... INTO OUTFILE or CREATE TABLE ... SELECT.

As of MySQL 5.0, you can use SELECT ... INTO with user-defined variables. The same syntax can also be used inside stored routines using cursors and local variables. See [Section 17.2.7.3, “SELECT ... INTO Statement”](#).

1.9.5.3. Transactions and Atomic Operations

MySQL Server (version 3.23-max and all versions 4.0 and above) supports transactions with the InnoDB and BDB transactional storage engines. InnoDB provides *full* ACID compliance. See [Chapter 14, Storage Engines and Table Types](#). For information about InnoDB differences from standard SQL with regard

to treatment of transaction errors, see [Section 14.2.15, “InnoDB Error Handling”](#).

The other non-transactional storage engines in MySQL Server (such as MyISAM) follow a different paradigm for data integrity called “atomic operations.” In transactional terms, MyISAM tables effectively always operate in AUTOCOMMIT=1 mode. Atomic operations often offer comparable integrity with higher performance.

Because MySQL Server supports both paradigms, you can decide whether your applications are best served by the speed of atomic operations or the use of transactional features. This choice can be made on a per-table basis.

As noted, the trade-off for transactional versus non-transactional storage engines lies mostly in performance. Transactional tables have significantly higher memory and disk space requirements, and more CPU overhead. On the other hand, transactional storage engines such as InnoDB also offer many significant features. MySQL Server's modular design allows the concurrent use of different storage engines to suit different requirements and deliver optimum performance in all situations.

But how do you use the features of MySQL Server to maintain rigorous integrity even with the non-transactional MyISAM tables, and how do these features compare with the transactional storage engines?

- If your applications are written in a way that is dependent on being able to call ROLLBACK rather than COMMIT in critical situations, transactions are more convenient. Transactions also ensure that unfinished updates or corrupting activities are not committed to the database; the server is given the opportunity to do an automatic rollback and your database is saved.

If you use non-transactional tables, MySQL Server in almost all cases allows you to resolve potential problems by including simple checks before updates and by running simple scripts that check the databases for inconsistencies and automatically repair or warn if such an inconsistency occurs. Note that just by using the MySQL log or even adding one extra log, you can normally fix tables perfectly with no data integrity loss.

- More often than not, critical transactional updates can be rewritten to be atomic. Generally speaking, all integrity problems that transactions solve can be done with LOCK TABLES or atomic updates, ensuring that there are no

automatic aborts from the server, which is a common problem with transactional database systems.

- To be safe with MySQL Server, regardless of whether you use transactional tables, you only need to have backups and have binary logging turned on. When that is true, you can recover from any situation that you could with any other transactional database system. It is always good to have backups, regardless of which database system you use.

The transactional paradigm has its benefits and its drawbacks. Many users and application developers depend on the ease with which they can code around problems where an abort appears to be necessary, or is necessary. However, even if you are new to the atomic operations paradigm, or more familiar with transactions, do consider the speed benefit that non-transactional tables can offer on the order of three to five times the speed of the fastest and most optimally tuned transactional tables.

In situations where integrity is of highest importance, MySQL Server offers transaction-level reliability and integrity even for non-transactional tables. If you lock tables with `LOCK TABLES`, all updates stall until integrity checks are made. If you obtain a `READ LOCAL` lock (as opposed to a write lock) for a table that allows concurrent inserts at the end of the table, reads are allowed, as are inserts by other clients. The newly inserted records are not be seen by the client that has the read lock until it releases the lock. With `INSERT DELAYED`, you can write inserts that go into a local queue until the locks are released, without having the client wait for the insert to complete. See [Section 7.3.3, “Concurrent Inserts”](#), and [Section 13.2.4.2, “INSERT DELAYED Syntax”](#).

“Atomic,” in the sense that we mean it, is nothing magical. It only means that you can be sure that while each specific update is running, no other user can interfere with it, and there can never be an automatic rollback (which can happen with transactional tables if you are not very careful). MySQL Server also guarantees that there are no dirty reads.

Following are some techniques for working with non-transactional tables:

- Loops that need transactions normally can be coded with the help of `LOCK TABLES`, and you don't need cursors to update records on the fly.
- To avoid using `ROLLBACK`, you can employ the following strategy:

1. Use `LOCK TABLES` to lock all the tables you want to access.
2. Test the conditions that must be true before performing the update.
3. Update if the conditions are satisfied.
4. Use `UNLOCK TABLES` to release your locks.

This is usually a much faster method than using transactions with possible rollbacks, although not always. The only situation this solution doesn't handle is when someone kills the threads in the middle of an update. In that case, all locks are released but some of the updates may not have been executed.

- You can also use functions to update records in a single operation. You can get a very efficient application by using the following techniques:
 - Modify columns relative to their current value.
 - Update only those columns that actually have changed.

For example, when we are updating customer information, we update only the customer data that has changed and test only that none of the changed data, or data that depends on the changed data, has changed compared to the original row. The test for changed data is done with the `WHERE` clause in the `UPDATE` statement. If the record wasn't updated, we give the client a message: "Some of the data you have changed has been changed by another user." Then we show the old row versus the new row in a window so that the user can decide which version of the customer record to use.

This gives us something that is similar to column locking but is actually even better because we only update some of the columns, using values that are relative to their current values. This means that typical `UPDATE` statements look something like these:

```
UPDATE tablename SET pay_back=pay_back+125;
```

```
UPDATE customer
  SET
    customer_date='current_date',
    address='new address',
```

```
    phone='new phone',
    money_owed_to_us=money_owed_to_us-125
WHERE
    customer_id=id AND address='old address' AND phone='old phor
```

This is very efficient and works even if another client has changed the values in the `pay_back` or `money_owed_to_us` columns.

- In many cases, users have wanted `LOCK TABLES` or `ROLLBACK` for the purpose of managing unique identifiers. This can be handled much more efficiently without locking or rolling back by using an `AUTO_INCREMENT` column and either the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. See [Section 12.9.3, “Information Functions”](#), and [Section 22.2.3.36, “mysql_insert_id\(\)”](#).

You can generally code around the need for row-level locking. Some situations really do need it, and InnoDB tables support row-level locking. Otherwise, with MyISAM tables, you can use a flag column in the table and do something like the following:

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID;
```

MySQL returns 1 for the number of affected rows if the row was found and `row_flag` wasn't 1 in the original row. You can think of this as though MySQL Server changed the preceding statement to:

```
UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
```

1.9.5.4. Stored Routines and Triggers

Stored procedures and functions are implemented beginning with MySQL 5.0. See [Chapter 17, *Stored Procedures and Functions*](#).

Basic trigger functionality is implemented beginning with MySQL 5.0.2, with further development planned for MySQL 5.1. See [Chapter 18, *Triggers*](#).

1.9.5.5. Foreign Keys

In MySQL Server 3.23.44 and up, the InnoDB storage engine supports checking of foreign key constraints, including `CASCADE`, `ON DELETE`, and `ON UPDATE`. See

[Section 14.2.6.4, “FOREIGN KEY Constraints”](#).

For storage engines other than InnoDB, MySQL Server parses the FOREIGN KEY syntax in CREATE TABLE statements, but does not use or store it. In the future, the implementation will be extended to store this information in the table specification file so that it may be retrieved by **mysqldump** and ODBC. At a later stage, foreign key constraints will be implemented for MyISAM tables as well.

Foreign key enforcement offers several benefits to database developers:

- Assuming proper design of the relationships, foreign key constraints make it more difficult for a programmer to introduce an inconsistency into the database.
- Centralized checking of constraints by the database server makes it unnecessary to perform these checks on the application side. This eliminates the possibility that different applications may not all check the constraints in the same way.
- Using cascading updates and deletes can simplify the application code.
- Properly designed foreign key rules aid in documenting relationships between tables.

Do keep in mind that these benefits come at the cost of additional overhead for the database server to perform the necessary checks. Additional checking by the server affects performance, which for some applications may be sufficiently undesirable as to be avoided if possible. (Some major commercial applications have coded the foreign key logic at the application level for this reason.)

MySQL gives database developers the choice of which approach to use. If you don't need foreign keys and want to avoid the overhead associated with enforcing referential integrity, you can choose another storage engine instead, such as MyISAM. (For example, the MyISAM storage engine offers very fast performance for applications that perform only INSERT and SELECT operations. In this case, the table has no holes in the middle and the inserts can be performed concurrently with retrievals. See [Section 7.3.3, “Concurrent Inserts”](#).)

If you choose not to take advantage of referential integrity checks, keep the

following considerations in mind:

- In the absence of server-side foreign key relationship checking, the application itself must handle relationship issues. For example, it must take care to insert rows into tables in the proper order, and to avoid creating orphaned child records. It must also be able to recover from errors that occur in the middle of multiple-record insert operations.
- If `ON DELETE` is the only referential integrity capability an application needs, you can achieve a similar effect as of MySQL Server 4.0 by using multiple-table `DELETE` statements to delete rows from many tables with a single statement. See [Section 13.2.1, “DELETE Syntax”](#).
- A workaround for the lack of `ON DELETE` is to add the appropriate `DELETE` statements to your application when you delete records from a table that has a foreign key. In practice, this is often as quick as using foreign keys and is more portable.

Be aware that the use of foreign keys can sometimes lead to problems:

- Foreign key support addresses many referential integrity issues, but it is still necessary to design key relationships carefully to avoid circular rules or incorrect combinations of cascading deletes.
- It is not uncommon for a DBA to create a topology of relationships that makes it difficult to restore individual tables from a backup. (MySQL alleviates this difficulty by allowing you to temporarily disable foreign key checks when reloading a table that depends on other tables. See [Section 14.2.6.4, “FOREIGN KEY Constraints”](#). As of MySQL 4.1.1, **mysqldump** generates dump files that take advantage of this capability automatically when they are reloaded.)

Note that foreign keys in SQL are used to check and enforce referential integrity, not to join tables. If you want to get results from multiple tables from a `SELECT` statement, you do this by performing a join between them:

```
SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

See [Section 13.2.7.1, “JOIN Syntax”](#), and [Section 3.6.6, “Using Foreign Keys”](#).

The FOREIGN KEY syntax without ON DELETE . . . is often used by ODBC applications to produce automatic WHERE clauses.

1.9.5.6. Views

Views (including updatable views) are implemented beginning with MySQL Server 5.0.1. See [Chapter 19, Views](#).

Views are useful for allowing users to access a set of relations (tables) as if it were a single table, and limiting their access to just that. Views can also be used to restrict access to rows (a subset of a particular table). For access control to columns, you can also use the sophisticated privilege system in MySQL Server. See [Section 5.8, “The MySQL Access Privilege System”](#).

In designing an implementation of views, our ambitious goal, as much as is possible within the confines of SQL, has been full compliance with “Codd's Rule #6” for relational database systems: “All views that are theoretically updatable, should in practice also be updatable.”

1.9.5.7. '-' as the Start of a Comment

Standard SQL uses the C syntax `/* this is a comment */` for comments, and MySQL Server supports this syntax as well. MySQL also support extensions to this syntax that allow MySQL-specific SQL to be embedded in the comment, as described in [Section 9.4, “Comment Syntax”](#).

Standard SQL uses `'--'` as a start-comment sequence. MySQL Server uses `'#'` as the start comment character. MySQL Server 3.23.3 and up also supports a variant of the `'--'` comment style. That is, the `'--'` start-comment sequence must be followed by a space (or by a control character such as a newline). The space is required to prevent problems with automatically generated SQL queries that use constructs such as the following, where we automatically insert the value of the payment for payment:

```
UPDATE account SET credit=credit-payment
```

Consider about what happens if payment has a negative value such as -1:

```
UPDATE account SET credit=credit--1
```

`credit--1` is a legal expression in SQL, but ‘--’ is interpreted as the start of a comment, part of the expression is discarded. The result is a statement that has a completely different meaning than intended:

```
UPDATE account SET credit=credit
```

The statement produces no change in value at all. This illustrates that allowing comments to start with ‘--’ can have serious consequences.

Using our implementation requires a space following the ‘--’ in order for it to be recognized as a start-comment sequence in MySQL Server 3.23.3 and newer. Therefore, `credit --1` is safe to use.

Another safe feature is that the **mysql** command-line client ignores lines that start with ‘--’.

The following information is relevant only if you are running a MySQL version earlier than 3.23.3:

If you have an SQL script in a text file that contains ‘--’ comments, you should use the **replace** utility as follows to convert the comments to use ‘#’ characters before executing the script:

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \  
      | mysql db_name
```

That is safer than executing the script in the usual way:

```
shell> mysql db_name < text-file-with-funny-comments.sql
```

You can also edit the script file “in place” to change the ‘--’ comments to ‘#’ comments:

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

Change them back with this command:

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

See [Section 8.18, “**replace** — A String-Replacement Utility”](#).

1.9.6. How MySQL Deals with Constraints

MySQL allows you to work both with transactional tables that allow rollback and with non-transactional tables that do not. Because of this, constraint handling is a bit different in MySQL than in other DBMSs. We must handle the case when you have inserted or updated a lot of rows in a non-transactional table for which changes cannot be rolled back when an error occurs.

The basic philosophy is that MySQL Server tries to produce an error for anything that it can detect while parsing a statement to be executed, and tries to recover from any errors that occur while executing the statement. We do this in most cases, but not yet for all.

The options MySQL has when an error occurs are to stop the statement in the middle or to recover as well as possible from the problem and continue. By default, the server follows the latter course. This means, for example, that the server may coerce illegal values to the closest legal values.

Beginning with MySQL 5.0.2, several SQL mode options are available to provide greater control over handling of bad data values and whether to continue statement execution or abort when errors occur. Using these options, you can configure MySQL Server to act in a more traditional fashion that is like other DBMSs that reject improper input. The SQL mode can be set globally at server startup to affect all clients. Individual clients can set the SQL mode at runtime, which enables each client to select the behavior most appropriate for its requirements. See [Section 5.2.5, “The Server SQL Mode”](#).

The following sections describe how MySQL Server handles different types of constraints.

1.9.6.1. PRIMARY KEY and UNIQUE Index Constraints

Normally, an error occurs when you try to INSERT or UPDATE a row that causes a primary key, unique key, or foreign key violation. If you are using a transactional storage engine such as InnoDB, MySQL automatically rolls back the statement. If you are using a non-transactional storage engine, MySQL stops processing the statement at the row for which the error occurred and leaves any remaining rows unprocessed.

If you want to ignore such key violations, MySQL supports an IGNORE keyword for INSERT and UPDATE. In this case, MySQL ignores any key violations and

continues processing with the next row. See [Section 13.2.4, “INSERT Syntax”](#), and [Section 13.2.10, “UPDATE Syntax”](#).

You can get information about the number of rows actually inserted or updated with the `mysql_info()` C API function. In MySQL 4.1 and up, you also can use the `SHOW WARNINGS` statement. See [Section 22.2.3.34, “mysql_info\(\)”](#), and [Section 13.5.4.25, “SHOW WARNINGS Syntax”](#).

Currently, only InnoDB tables support foreign keys. See [Section 14.2.6.4, “FOREIGN KEY Constraints”](#). Foreign key support in MyISAM tables is scheduled for implementation in MySQL 5.2. See [Section 1.6, “MySQL Development Roadmap”](#).

1.9.6.2. Constraints on Invalid Data

Before MySQL 5.0.2, MySQL is forgiving of illegal or improper data values and coerces them to legal values for data entry. In MySQL 5.0.2 and up, that remains the default behavior, but you can change the server SQL mode to select more traditional treatment of bad values such that the server rejects them and aborts the statement in which they occur. [Section 5.2.5, “The Server SQL Mode”](#).

This section describes the default (forgiving) behavior of MySQL, as well as the newer strict SQL mode and how it differs.

If you are not using strict mode, then whenever you insert an “incorrect” value into a column, such as a `NULL` into a `NOT NULL` column or a too-large numeric value into a numeric column, MySQL sets the column to the “best possible value” instead of producing an error: The following rules describe in more detail how this works:

- If you try to store an out of range value into a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value, whichever is closest to the invalid value.
- For strings, MySQL stores either the empty string or as much of the string as can be stored in the column.
- If you try to store a string that doesn't start with a number into a numeric column, MySQL Server stores 0.

- Invalid values for ENUM and SET columns are handled as described in [Section 1.9.6.3, “ENUM and SET Constraints”](#).
- MySQL allows you to store certain incorrect date values into DATE and DATETIME columns (such as '2000-02-31' or '2000-02-00'). The idea is that it's not the job of the SQL server to validate dates. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special “zero” date value '0000-00-00' is stored in the column instead.
- If you try to store NULL into a column that doesn't take NULL values, an error occurs for single-row INSERT statements. For multiple-row INSERT statements or for INSERT INTO ... SELECT statements, MySQL Server stores the implicit default value for the column data type. In general, this is 0 for numeric types, the empty string (' ') for string types, and the “zero” value for date and time types. Implicit default values are discussed in [Section 11.1.4, “Data Type Default Values”](#).
- If an INSERT statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit DEFAULT clause. If the definition has no such DEFAULT clause, MySQL inserts the implicit default value for the column data type.

The reason for using the preceding rules in non-strict mode is that we can't check these conditions until the statement has begun executing. We can't just roll back if we encounter a problem after updating a few rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in this case, the update would be “half done,” which is probably the worst possible scenario. In this case, it's better to “do the best you can” and then continue as if nothing happened.

In MySQL 5.0.2 and up, you can select stricter treatment of input values by using the STRICT_TRANS_TABLES or STRICT_ALL_TABLES SQL modes:

```
SET sql_mode = 'STRICT_TRANS_TABLES';
SET sql_mode = 'STRICT_ALL_TABLES';
```

STRICT_TRANS_TABLES enables strict mode for transactional storage engines, and also to some extent for non-transactional engines. It works like this:

- For transactional storage engines, bad data values occurring anywhere in a statement cause the statement to abort and roll back.
- For non-transactional storage engines, a statement aborts if the error occurs in the first row to be inserted or updated. (When the error occurs in the first row, the statement can be aborted to leave the table unchanged, just as for a transactional table.) Errors in rows after the first do not abort the statement, because the table has already been changed by the first row. Instead, bad data values are adjusted and result in warnings rather than errors. In other words, with `STRICT_TRANS_TABLES`, a wrong value causes MySQL to roll back all updates done so far, if that can be done without changing the table. But once the table has been changed, further errors result in adjustments and warnings.

For even stricter checking, enable `STRICT_ALL_TABLES`. This is the same as `STRICT_TRANS_TABLES` except that for non-transactional storage engines, errors abort the statement even for bad data in rows following the first row. This means that if an error occurs partway through a multiple-row insert or update for a non-transactional table, a partial update results. Earlier rows are inserted or updated, but those from the point of the error on are not. To avoid this for non-transactional tables, either use single-row statements or else use `STRICT_TRANS_TABLES` if conversion warnings rather than errors are acceptable. To avoid problems in the first place, do not use MySQL to check column content. It is safest (and often faster) to let the application ensure that it passes only legal values to the database.

With either of the strict mode options, you can cause errors to be treated as warnings by using `INSERT IGNORE` or `UPDATE IGNORE` rather than `INSERT` or `UPDATE` without `IGNORE`.

1.9.6.3. ENUM and SET Constraints

ENUM and SET columns provide an efficient way to define columns that can contain only a given set of values. See [Section 11.4.4, “The ENUM Type”](#), and [Section 11.4.5, “The SET Type”](#). However, before MySQL 5.0.2, ENUM and SET columns do not provide true constraints on entry of invalid data:

- ENUM columns always have a default value. If you specify no default value, then it is NULL for columns that can have NULL, otherwise it is the first

enumeration value in the column definition.

- If you insert an incorrect value into an ENUM column or if you force a value into an ENUM column with IGNORE, it is set to the reserved enumeration value of 0, which is displayed as an empty string in string context.
- If you insert an incorrect value into a SET column, the incorrect value is ignored. For example, if the column can contain the values 'a', 'b', and 'c', an attempt to assign 'a, x, b, y' results in a value of 'a, b'.

As of MySQL 5.0.2, you can configure the server to use strict SQL mode. See [Section 5.2.5, “The Server SQL Mode”](#). With strict mode enabled, the definition of a ENUM or SET column does act as a constraint on values entered into the column. An error occurs for values that do not satisfy these conditions:

- An ENUM value must be one of those listed in the column definition, or the internal numeric equivalent thereof. The value cannot be the error value (that is, 0 or the empty string). For a column defined as ENUM('a', 'b', 'c'), values such as '', 'd', or 'ax' are illegal and are rejected.
- A SET value must be the empty string or a value consisting only of the values listed in the column definition separated by commas. For a column defined as SET('a', 'b', 'c'), values such as 'd' or 'a, b, c, d' are illegal and are rejected.

Errors for invalid values can be suppressed in strict mode if you use INSERT IGNORE or UPDATE IGNORE. In this case, a warning is generated rather than an error. For ENUM, the value is inserted as the error member (0). For SET, the value is inserted as given except that any invalid substrings are deleted. For example, 'a, x, b, y' results in a value of 'a, b'.

Chapter 2. Installing and Upgrading MySQL

Table of Contents

[2.1. General Installation Issues](#)

[2.1.1. Operating Systems Supported by MySQL](#)

[2.1.2. Choosing Which MySQL Distribution to Install](#)

[2.1.3. How to Get MySQL](#)

[2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG](#)

[2.1.5. Installation Layouts](#)

[2.2. Standard MySQL Installation Using a Binary Distribution](#)

[2.3. Installing MySQL on Windows](#)

[2.3.1. Choosing An Installation Package](#)

[2.3.2. Installing MySQL with the Automated Installer](#)

[2.3.3. Using the MySQL Installation Wizard](#)

[2.3.4. Using the Configuration Wizard](#)

[2.3.5. Installing MySQL from a Noinstall Zip Archive](#)

[2.3.6. Extracting the Install Archive](#)

[2.3.7. Creating an Option File](#)

[2.3.8. Selecting a MySQL Server type](#)

[2.3.9. Starting the Server for the First Time](#)

[2.3.10. Starting MySQL from the Windows Command Line](#)

[2.3.11. Starting MySQL as a Windows Service](#)

[2.3.12. Testing The MySQL Installation](#)

[2.3.13. Troubleshooting a MySQL Installation Under Windows](#)

[2.3.14. Upgrading MySQL on Windows](#)

[2.3.15. MySQL on Windows Compared to MySQL on Unix](#)

[2.4. Installing MySQL on Linux](#)

[2.5. Installing MySQL on Mac OS X](#)

[2.6. Installing MySQL on Solaris](#)

[2.7. Installing MySQL on NetWare](#)

[2.8. Installing MySQL on Other Unix-Like Systems](#)

[2.9. MySQL Installation Using a Source Distribution](#)

[2.9.1. Source Installation Overview](#)

[2.9.2. Typical **configure** Options](#)

[2.9.3. Installing from the Development Source Tree](#)

[2.9.4. Dealing with Problems Compiling MySQL](#)

- [2.9.5. MIT-pthreads Notes](#)
- [2.9.6. Installing MySQL from Source on Windows](#)
- [2.9.7. Compiling MySQL Clients on Windows](#)
- [2.10. Post-Installation Setup and Testing](#)
 - [2.10.1. Windows Post-Installation Procedures](#)
 - [2.10.2. Unix Post-Installation Procedures](#)
 - [2.10.3. Securing the Initial MySQL Accounts](#)
- [2.11. Upgrading MySQL](#)
 - [2.11.1. Upgrading from MySQL 5.0 to 5.1](#)
 - [2.11.2. Upgrading from MySQL 4.1 to 5.0](#)
 - [2.11.3. Copying MySQL Databases to Another Machine](#)
- [2.12. Downgrading MySQL](#)
 - [2.12.1. Downgrading to MySQL 4.1](#)
- [2.13. Operating System-Specific Notes](#)
 - [2.13.1. Linux Notes](#)
 - [2.13.2. Mac OS X Notes](#)
 - [2.13.3. Solaris Notes](#)
 - [2.13.4. BSD Notes](#)
 - [2.13.5. Other Unix Notes](#)
 - [2.13.6. OS/2 Notes](#)
- [2.14. Perl Installation Notes](#)
 - [2.14.1. Installing Perl on Unix](#)
 - [2.14.2. Installing ActiveState Perl on Windows](#)
 - [2.14.3. Problems Using the Perl DBI/DBD Interface](#)

This chapter describes how to obtain and install MySQL. A summary of the procedure follows and later sections provide the details. If you plan to upgrade an existing version of MySQL to a newer version rather than install MySQL for the first time, see [Section 2.11, “Upgrading MySQL”](#), for information about upgrade procedures and about issues that you should consider before upgrading.

1. **Determine whether your platform is supported.** Please note that not all supported systems are equally suitable for running MySQL. On some platforms it is much more robust and efficient than others. See [Section 2.1.1, “Operating Systems Supported by MySQL”](#), for details.
2. **Choose which distribution to install.** Several versions of MySQL are available, and most are available in several distribution formats. You can choose from pre-packaged distributions containing binary (precompiled)

programs or source code. When in doubt, use a binary distribution. We also provide public access to our current source tree for those who want to see our most recent developments and help us test new code. To determine which version and type of distribution you should use, see [Section 2.1.2, “Choosing Which MySQL Distribution to Install”](#).

3. **Download the distribution that you want to install.** For instructions, see [Section 2.1.3, “How to Get MySQL”](#). To verify the integrity of the distribution, use the instructions in [Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”](#).
4. **Install the distribution.** To install MySQL from a binary distribution, use the instructions in [Section 2.2, “Standard MySQL Installation Using a Binary Distribution”](#). To install MySQL from a source distribution or from the current development source tree, use the instructions in [Section 2.9, “MySQL Installation Using a Source Distribution”](#).

If you encounter installation difficulties, see [Section 2.13, “Operating System-Specific Notes”](#), for information on solving problems for particular platforms.

5. **Perform any necessary post-installation setup.** After installing MySQL, read [Section 2.10, “Post-Installation Setup and Testing”](#). This section contains important information about making sure the MySQL server is working properly. It also describes how to secure the initial MySQL user accounts, *which have no passwords* until you assign passwords. The section applies whether you install MySQL using a binary or source distribution.
6. If you want to run the MySQL benchmark scripts, Perl support for MySQL must be available. See [Section 2.14, “Perl Installation Notes”](#).

2.1. General Installation Issues

Before installing MySQL, you should do the following:

1. Determine whether MySQL runs on your platform.
2. Choose a distribution to install.
3. Download the distribution and verify its integrity.

This section contains the information necessary to carry out these steps. After doing so, you can use the instructions in later sections of the chapter to install the distribution that you choose.

2.1.1. Operating Systems Supported by MySQL

This section lists the operating systems on which you can expect to be able to run MySQL.

We use GNU Autoconf, so it is possible to port MySQL to all modern systems that have a C++ compiler and a working implementation of POSIX threads. (Thread support is needed for the server. To compile only the client code, the only requirement is a C++ compiler.) We use and develop the software ourselves primarily on Linux (SuSE and Red Hat), FreeBSD, and Sun Solaris (versions 8 and 9).

MySQL has been reported to compile successfully on the following combinations of operating system and thread package. Note that for many operating systems, native thread support works only in the latest versions.

- AIX 4.x, 5.x with native threads. See [Section 2.13.5.3, “IBM-AIX notes”](#).
- Amiga.
- BSDI 2.x with the MIT-pthreads package. See [Section 2.13.4.4, “BSD/OS Version 2.x Notes”](#).
- BSDI 3.0, 3.1 and 4.x with native threads. See [Section 2.13.4.4, “BSD/OS](#)

[Version 2.x Notes](#)”.

- Digital Unix 4.x with native threads. See [Section 2.13.5.5, “Alpha-DEC-UNIX Notes \(Tru64\)”](#).
- FreeBSD 2.x with the MIT-pthreads package. See [Section 2.13.4.1, “FreeBSD Notes”](#).
- FreeBSD 3.x and 4.x with native threads. See [Section 2.13.4.1, “FreeBSD Notes”](#).
- FreeBSD 4.x with LinuxThreads. See [Section 2.13.4.1, “FreeBSD Notes”](#).
- HP-UX 10.20 with the DCE threads or the MIT-pthreads package. See [Section 2.13.5.1, “HP-UX Version 10.20 Notes”](#).
- HP-UX 11.x with the native threads. See [Section 2.13.5.2, “HP-UX Version 11.x Notes”](#).
- Linux 2.0+ with LinuxThreads 0.7.1+ or glibc 2.0.7+ for various CPU architectures. See [Section 2.13.1, “Linux Notes”](#).
- Mac OS X. See [Section 2.13.2, “Mac OS X Notes”](#).
- NetBSD 1.3/1.4 Intel and NetBSD 1.3 Alpha (requires GNU make). See [Section 2.13.4.2, “NetBSD Notes”](#).
- Novell NetWare 6.0 and 6.5. See [Section 2.7, “Installing MySQL on NetWare”](#).
- OpenBSD 2.5 and with native threads. OpenBSD earlier than 2.5 with the MIT-pthreads package. See [Section 2.13.4.3, “OpenBSD 2.5 Notes”](#).
- OS/2 Warp 3, FixPack 29 and OS/2 Warp 4, FixPack 4. See [Section 2.13.6, “OS/2 Notes”](#).
- SCO OpenServer 5.0.X with a recent port of the FSU Pthreads package. See [Section 2.13.5.8, “SCO UNIX and OpenServer 5.0.x Notes”](#).
- SCO Openserver 6.0.x. See [Section 2.13.5.9, “SCO OpenServer 6.0.x](#)

[Notes](#)".

- SCO UnixWare 7.1.x. See [Section 2.13.5.10, "SCO UnixWare 7.1.x and OpenUNIX 8.0.0 Notes"](#).
- SGI Irix 6.x with native threads. See [Section 2.13.5.7, "SGI Irix Notes"](#).
- Solaris 2.5 and above with native threads on SPARC and x86. See [Section 2.13.3, "Solaris Notes"](#).
- SunOS 4.x with the MIT-pthreads package. See [Section 2.13.3, "Solaris Notes"](#).
- Tru64 Unix. See [Section 2.13.5.5, "Alpha-DEC-UNIX Notes \(Tru64\)"](#).
- Windows 9x, Me, NT, 2000, XP, and Windows Server 2003. See [Section 2.3, "Installing MySQL on Windows"](#).

Not all platforms are equally well-suited for running MySQL. How well a certain platform is suited for a high-load mission-critical MySQL server is determined by the following factors:

- General stability of the thread library. A platform may have an excellent reputation otherwise, but MySQL is only as stable as the thread library it calls, even if everything else is perfect.
- The capability of the kernel and the thread library to take advantage of symmetric multi-processor (SMP) systems. In other words, when a process creates a thread, it should be possible for that thread to run on a CPU different from the original process.
- The capability of the kernel and the thread library to run many threads that acquire and release a mutex over a short critical region frequently without excessive context switches. If the implementation of `pthread_mutex_lock()` is too anxious to yield CPU time, this hurts MySQL tremendously. If this issue is not taken care of, adding extra CPUs actually makes MySQL slower.
- General filesystem stability and performance.

- If your tables are large, performance is affected by the ability of the filesystem to deal with large files at all and to deal with them efficiently.
- Our level of expertise here at MySQL AB with the platform. If we know a platform well, we enable platform-specific optimizations and fixes at compile time. We can also provide advice on configuring your system optimally for MySQL.
- The amount of testing we have done internally for similar configurations.
- The number of users that have run MySQL successfully on the platform in similar configurations. If this number is high, the likelihood of encountering platform-specific surprises is much smaller.

Based on the preceding criteria, the best platforms for running MySQL at this point are x86 with SuSE Linux using a 2.4 or 2.6 kernel, and ReiserFS (or any similar Linux distribution) and SPARC with Solaris (2.7-9). FreeBSD comes third, but we really hope it joins the top club once the thread library is improved. We also hope that at some point we are able to include into the top category all other platforms on which MySQL currently compiles and runs, but not quite with the same level of stability and performance. This requires some effort on our part in cooperation with the developers of the operating systems and library components that MySQL depends on. If you are interested in improving one of those components, are in a position to influence its development, and need more detailed instructions on what MySQL needs to run better, send an email message to the MySQL `internals` mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#).

Please note that the purpose of the preceding comparison is not to say that one operating system is better or worse than another in general. We are talking only about choosing an OS for the specific purpose of running MySQL. With this in mind, the result of this comparison might be different if other factors were considered. In some cases, the reason one OS is better for MySQL than another might simply be that we have been able to put more effort into testing and optimizing for a particular platform. We are just stating our observations to help you decide which platform to use for running MySQL.

2.1.2. Choosing Which MySQL Distribution to Install

When preparing to install MySQL, you should decide which version to use. MySQL development occurs in several release series, and you can pick the one that best fits your needs. After deciding which version to install, you can choose a distribution format. Releases are available in binary or source format.

2.1.2.1. Choosing Which Version of MySQL to Install

The first decision to make is whether you want to use a production (stable) release or a development release. In the MySQL development process, multiple release series co-exist, each at a different stage of maturity:

- MySQL 5.1 is the current development release series.
- MySQL 5.0 is the current stable (production-quality) release series. New releases are issued for bugfixes only; no new features are being added that could effect stability.
- MySQL 4.1 is the previous stable (production-quality) release series. New releases are issued for critical bugfixes and security fixes. No significant new features are to be added to this series.
- MySQL 4.0 and 3.23 are the old stable (production-quality) release series. These versions are now retired, so new releases are issued only to fix extremely critical bugs (primarily security issues).

We do not believe in a complete code freeze because this prevents us from making bugfixes and other fixes that must be done. By “somewhat frozen” we mean that we may add small things that should not affect anything that currently works in a production release. Naturally, relevant bugfixes from an earlier series propagate to later series.

Normally, if you are beginning to use MySQL for the first time or trying to port it to some system for which there is no binary distribution, we recommend going with the production release series. Currently, this is MySQL 5.0. All MySQL releases, even those from development series, are checked with the MySQL benchmarks and an extensive test suite before being issued.

If you are running an older system and want to upgrade, but do not want to take the chance of having a non-seamless upgrade, you should upgrade to the latest

version in the same release series you are using (where only the last part of the version number is newer than yours). We have tried to fix only fatal bugs and make only small, relatively “safe” changes to that version.

If you want to use new features not present in the production release series, you can use a version from a development series. Note that development releases are not as stable as production releases.

If you want to use the very latest sources containing all current patches and bugfixes, you can use one of our BitKeeper repositories. These are not “releases” as such, but are available as previews of the code on which future releases are to be based.

The MySQL naming scheme uses release names that consist of three numbers and a suffix; for example, **mysql-5.0.12-beta**. The numbers within the release name are interpreted as follows:

- The first number (**5**) is the major version and describes the file format. All MySQL 5 releases have the same file format.
- The second number (**0**) is the release level. Taken together, the major version and release level constitute the release series number.
- The third number (**12**) is the version number within the release series. This is incremented for each new release. Usually you want the latest version for the series you have chosen.

For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.

Release names also include a suffix to indicate the stability level of the release. Releases within a series progress through a set of suffixes to indicate how the stability level improves. The possible suffixes are:

- **alpha** indicates that the release contains new features that have not been thoroughly tested. Known bugs should be documented in the News section. See [Appendix D, MySQL Change History](#). Most alpha releases implement new commands and extensions. Active development that may involve

major code changes can occur in an alpha release. However, we do conduct testing before issuing a release.

- **beta** means that the release is intended to be feature-complete and that all new code has been tested. No major new features that are added. There should be no known critical bugs. A version changes from alpha to beta when there have been no reported fatal bugs within an alpha version for at least a month and we have no plans to add any new features that could make previously implemented features unreliable.

All APIs, externally visible structures, and columns for SQL statements will not change during future beta, release candidate, or production releases.

- **rc** is a release candidate; that is, a beta that has been around for a while and seems to work well. Only minor fixes are added. (A release candidate is what formerly was known as a gamma release.)
- If there is no suffix, it means that the version has been run for a while at many different sites with no reports of critical repeatable bugs other than platform-specific bugs. Only critical bugfixes are applied to the release. This is what we call a production (stable) or “General Availability” (GA) release.

MySQL uses a naming scheme that is slightly different from most other products. In general, it is usually safe to use any version that has been out for a couple of weeks without being replaced by a new version within the same release series.

All releases of MySQL are run through our standard tests and benchmarks to ensure that they are relatively safe to use. Because the standard tests are extended over time to check for all previously found bugs, the test suite keeps getting better.

All releases have been tested at least with these tools:

- An internal test suite

The `mysql-test` directory contains an extensive set of test cases. We run these tests for virtually every server binary. See [Section 24.1.2, “MySQL Test Suite”](#), for more information about this test suite.

- The MySQL benchmark suite

This suite runs a range of common queries. It is also a test to determine whether the latest batch of optimizations actually made the code faster. See [Section 7.1.4, “The MySQL Benchmark Suite”](#).

- The crash-me test

This test tries to determine what features the database supports and what its capabilities and limitations are. See [Section 7.1.4, “The MySQL Benchmark Suite”](#).

We also test the newest MySQL version in our internal production environment, on at least one machine. We have more than 100GB of data to work with.

2.1.2.2. Choosing a Distribution Format

After choosing which version of MySQL to install, you should decide whether to use a binary distribution or a source distribution. In most cases, you should probably use a binary distribution, if one exists for your platform. Binary distributions are available in native format for many platforms, such as RPM files for Linux or PKG package installers for Mac OS X or Solaris. Distributions also are available as Zip archives or compressed **tar** files.

Reasons to choose a binary distribution include the following:

- Binary distributions generally are easier to install than source distributions.
- To satisfy different user requirements, we provide two different binary versions. One is compiled with the core feature set. The other (MySQL-Max) is compiled with an extended feature set. Both versions are compiled from the same source distribution. All native MySQL clients can connect to servers from either MySQL version.

The extended MySQL binary distribution is identified by the `-max` suffix and is configured with the same options as **mysqld-max**. See [Section 5.3, “The **mysqld-max** Extended MySQL Server”](#).

For RPM distributions, if you want to use the MySQL-Max RPM, you must

first install the standard MySQL-server RPM.

Under some circumstances, you may be better off installing MySQL from a source distribution:

- You want to install MySQL at some explicit location. The standard binary distributions are ready to run at any installation location, but you might require even more flexibility to place MySQL components where you want.
- You want to configure **mysqld** to ensure that features are available that might not be included in the standard binary distributions. Here is a list of the most common extra options that you may want to use to ensure feature availability:
 - `--with-innodb`
 - `--with-berkeley-db` (not available on all platforms)
 - `--with-libwrap`
 - `--with-named-z-libs` (this is done for some of the binaries)
 - `--with-debug[=full]`
- You want to configure **mysqld** without some features that are included in the standard binary distributions. For example, distributions normally are compiled with support for all character sets. If you want a smaller MySQL server, you can recompile it with support for only the character sets you need.
- You have a special compiler (such as `pgcc`) or want to use compiler options that are better optimized for your processor. Binary distributions are compiled with options that should work on a variety of processors from the same processor family.
- You want to use the latest sources from one of the BitKeeper repositories to have access to all current bugfixes. For example, if you have found a bug and reported it to the MySQL development team, the bugfix is committed to the source repository and you can access it there. The bugfix does not appear in a release until a release actually is issued.

- You want to read (or modify) the C and C++ code that makes up MySQL. For this purpose, you should get a source distribution, because the source code is always the ultimate manual.
- Source distributions contain more tests and examples than binary distributions.

2.1.2.3. How and When Updates Are Released

MySQL is evolving quite rapidly and we want to share new developments with other MySQL users. We try to produce a new release whenever we have new and useful features that others also seem to have a need for.

We also try to help users who request features that are easy to implement. We take note of what our licensed users want, and we especially take note of what our support customers want and try to help them in this regard.

No one is *required* to download a new release. The News section helps you determine whether the new release has something you really want. See [Appendix D, MySQL Change History](#).

We use the following policy when updating MySQL:

- Releases are issued within each series. For each release, the last number in the version is one more than the previous release within the same series.
- Production (stable) releases are meant to appear about 1-2 times a year. However, if small bugs are found, a release with only bugfixes is issued.
- Working releases/bugfixes to old releases are meant to appear about every 4-8 weeks.
- Binary distributions for some platforms are made by us for major releases. Other people may make binary distributions for other systems, but probably less frequently.
- We make fixes available as soon as we have identified and corrected small or non-critical but annoying bugs. The fixes are available immediately from our public BitKeeper repositories, and are included in the next release.

- If by any chance a fatal bug is found in a release, our policy is to fix it in a new release as soon as possible. (We would like other companies to do this, too!)

2.1.2.4. Release Philosophy—No Known Bugs in Releases

We put considerable time and effort into making our releases bug-free. Our policy is never to release a version of MySQL intended for production use that has any known fatal, repeatable bugs.

We have documented all open problems, bugs, and issues that are dependent on design decisions. See [Section A.8, “Known Issues in MySQL”](#).

Our aim is to fix everything that is fixable without making a stable MySQL version less stable. In certain cases, this means we can fix an issue in the development versions, but not in the stable (production) version. Naturally, we document such issues so that users are aware of them.

Here is a description of our build process:

- We monitor bugs from our customer support list, the bugs database at <http://bugs.mysql.com/>, and the MySQL external mailing lists.
- All reported bugs for live versions are entered into the bugs database.
- When we fix a bug, we always try to make a test case for it and include it into our test system to ensure that the bug can never recur without being detected. (About 90% of all fixed bugs have test cases.)
- We create test cases for each new feature that we add to MySQL.
- Before we start to build a new MySQL release, we ensure that all reported repeatable bugs for that MySQL version (3.23.x, 4.0.x, 4.1.x, 5.0.x, 5.1.x, and so on) are fixed. If something is impossible to fix due to some internal design decision in MySQL, we document this in the manual. See [Section A.8, “Known Issues in MySQL”](#).
- We do a build on all platforms for which we support binaries and run our test suite and benchmark suite on all of them.

- We do not publish a binary for a platform for which the test or benchmark suite fails. If the problem is due to a general error in the source, we fix it and do the build plus tests on all systems again from scratch.
- The build and test process takes a week. If we receive a report regarding a fatal bug during this process (for example, one that causes a core dump), we fix the problem and restart the build process.
- After publishing the binaries on <http://dev.mysql.com/>, we send out an announcement message to the mysql and announce mailing lists. See [Section 1.7.1, “MySQL Mailing Lists”](#). The announcement message contains a list of all changes to the release and any known problems with the release. The **Known Problems** section in the release notes has been needed for only a handful of releases.
- To quickly give our users access to the latest MySQL features, we try to produce a new MySQL release every 4-8 weeks. Source code snapshots are built daily and are available at <http://downloads.mysql.com/snapshots.php>.
- If, despite our best efforts, we receive any bug reports after a release is issued that a critical problem exists for the build on a specific platform, we fix it at once and build a new 'a' release for that platform. Thanks to our large user base, problems are found and resolved very quickly.
- Our track record for making stable releases is quite good. In the last 150 releases, we had to do a new build for fewer than 10 of them. In three of these cases, the bug was a faulty glibc library on one of our build machines that took us a long time to track down.

2.1.2.5. MySQL Binaries Compiled by MySQL AB

As a service of MySQL AB, we provide a set of binary distributions of MySQL that are compiled on systems at our site or on systems where supporters of MySQL kindly have given us access to their machines.

In addition to the binaries provided in platform-specific package formats, we offer binary distributions for a number of platforms in the form of compressed **tar** files (.tar.gz files). See [Section 2.2, “Standard MySQL Installation Using a Binary Distribution”](#).

The RPM distributions for MySQL 5.0 releases that we make available through our Web site are generated by MySQL AB.

For Windows distributions, see [Section 2.3, “Installing MySQL on Windows”](#).

These distributions are generated using the script `Build-tools/Do-compile`, which compiles the source code and creates the binary `tar.gz` archive using **`scripts/make_binary_distribution`**.

These binaries are configured and built with the following compilers and options. This information can also be obtained by looking at the variables `COMP_ENV_INFO` and `CONFIGURE_LINE` inside the script **`bin/mysqlbug`** of every binary **tar** file distribution.

Anyone who has more optimal options for any of the following configure commands can mail them to the MySQL internals mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#).

If you want to compile a debug version of MySQL, you should add `--with-debug` or `--with-debug=full` to the following **configure** commands and remove any `-fomit-frame-pointer` options.

The following binaries are built on MySQL AB development systems:

- Linux 2.4.xx x86 with **gcc** 2.95.3:

```
CFLAGS="-O2 -mcpu=pentiumpro" CXX=gcc CXXFLAGS="-O2 -mcpu=pentiu
-felide-constructors" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-asm --disable-shared
--with-client-ldflags=-all-static --with-mysqld-ldflags=-all-sta
```

- Linux 2.4.x x86 with **icc** (Intel C++ Compiler 8.1 or later releases):

```
CC=icc CXX=icpc CFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict"
CXXFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict" ./configure
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --enable-assen
--disable-shared --with-client-ldflags=-all-static
--with-mysqld-ldflags=-all-static --with-embedded-server --with-
```

Note that versions 8.1 and newer of the Intel compiler have separate drivers

for 'pure' C (icc) and C++ (icpc); if you use **icc** version 8.0 or older for building MySQL, you will need to set `CXX=icc`.

- Linux 2.4.xx Intel Itanium 2 with **ecc** (Intel C++ Itanium Compiler 7.0):

```
CC=ecc CFLAGS="-O2 -tpp2 -ip -nolib_inline" CXX=ecc CXXFLAGS="-C
-tpp2 -ip -nolib_inline" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile
```

- Linux 2.4.xx Intel Itanium with **ecc** (Intel C++ Itanium Compiler 7.0):

```
CC=ecc CFLAGS=-tpp1 CXX=ecc CXXFLAGS=-tpp1 ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile
```

- Linux 2.4.xx alpha with **ccc** (Compaq C V6.2-505 / Compaq C++ V6.3-006):

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx CXXFLAGS="-fast -arc
generic -noexceptions -nortti" ./configure --prefix=/usr/local/m
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-mysqld-ldflags=-non_shared
--with-client-ldflags=-non_shared --disable-shared
```

- Linux 2.x.xx ppc with **gcc** 2.95.4:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-C
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/my
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-embedded-server
--with-innodb
```

- Linux 2.4.xx s390 with **gcc** 2.95.3:

```
CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors" ./confi
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shar
--with-client-ldflags=-all-static --with-mysqld-ldflags=-all-sta
```

- Linux 2.4.xx x86_64 (AMD64) with **gcc** 3.2.1:

```
CXX=gcc ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
```

```
--enable-local-infile --disable-shared
```

- Sun Solaris 8 x86 with **gcc 3.2.3**:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --disable-shared --with-innodb
```

- Sun Solaris 8 SPARC with **gcc 3.2**:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --enable-assembler --with-named-z-libs=no --with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 8 SPARC 64-bit with **gcc 3.2**:

```
CC=gcc CFLAGS="-O3 -m64 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -m64 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --with-named-z-libs=no --with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 9 SPARC with **gcc 2.95.3**:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --enable-assembler --with-named-curses-lib --disable-shared
```

- Sun Solaris 9 SPARC with **cc-5.0 (Sun Forte 5.0)**:

```
CC=cc-5.0 CXX=CC ASFLAGS="-xarch=v9" CFLAGS="-Xa -xstrconst -mt -D_FORTEC_ -xarch=v9" CXXFLAGS="-noex -mt -D_FORTEC_ -xarch=v9" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --enable-assembly --with-named-z-libs=no --enable-thread-safe-client --disable-shared
```

- IBM AIX 4.3.2 ppc with **gcc 3.2.3**:

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many " CXX=gcc CXXFLAGS="-O2
-mcpu=powerpc -Wa,-many -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --disable-shared
```

- IBM AIX 4.3.3 ppc with x1c_r (IBM Visual Age C/C++ 6.0):

```
CC=x1c_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
CXX=x1c_r CXXFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z
--disable-shared --with-innodb
```

- IBM AIX 5.1.0 ppc with gcc 3.3:

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many" CXX=gcc CXXFLAGS="-O2 -mcpu
-Wa,-many -felide-constructors -fno-exceptions -fno-rtti" ./conf
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z
--disable-shared
```

- IBM AIX 5.2.0 ppc with x1c_r (IBM Visual Age C/C++ 6.0):

```
CC=x1c_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
CXX=x1c_r CXXFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --with-named-z
--disable-shared --with-embedded-server --with-innodb
```

- HP-UX 10.20 pa-risc1.1 with gcc 3.1:

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc CXXFLAGS="-
-I/opt/dce /include -felide-constructors -fno-exceptions -fno-rt
-O3 -fPIC" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-pthread --with-named-thread-libs=-l
--with-lib-ccflags=-fPIC --disable-shared
```

- HP-UX 11.00 pa-risc with acc (HP ANSI C++ B3910B A.03.50):

```
CC=cc CXX=aCC CFLAGS=+DAportable CXXFLAGS=+DAportable ./configur
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shar
```

--with-embedded-server --with-innodb

- HP-UX 11.11 pa-risc2.0 64bit with acc (HP ANSI C++ B3910B A.03.33):

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shar
```

- HP-UX 11.11 pa-risc2.0 32bit with acc (HP ANSI C++ B3910B A.03.33):

```
CC=cc CXX=aCC CFLAGS="+DAportable" CXXFLAGS="+DAportable" ./conf
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shar
--with-innodb
```

- HP-UX 11.22 ia64 64bit with acc (HP aC++/ANSI C B3910B A.05.50):

```
CC=cc CXX=aCC CFLAGS="+DD64 +DSitanium2" CXXFLAGS="+DD64 +DSitar
./configure --prefix=/usr/local/mysql --localstatedir=/usr/local
--libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile --disable-shar
--with-embedded-server --with-innodb
```

- Apple Mac OS X 10.2 powerpc with **gcc** 3.1:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-C
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

- FreeBSD 4.7 i386 with **gcc** 2.95.4:

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/my
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-asm --with-named-z-libs=not
--disable-shared
```

- FreeBSD 4.7 i386 using LinuxThreads with **gcc** 2.95.4:

```
CFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT
-D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads"
CXXFLAGS="-DHAVE_BROKEN_REALPATH -D__USE_UNIX98 -D_REENTRANT
-D_THREAD_SAFE -I/usr/local/include/pthread/linuxthreads" ./conf
--prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data
--libexecdir=/usr/local/mysql/bin --enable-thread-safe-client
```

```
--enable-local-infile --enable-assembler
--with-named-thread-libs="-DHAVE_GLIBC2_STYLE_GETHOSTBYNAME_R
-D_THREAD_SAFE -I /usr/local/include/pthread/linuxthreads
-L/usr/local/lib -llthread -llgcc_r" --disable-shared
--with-embedded-server --with-innodb
```

- QNX Neutrino 6.2.1 i386 with **gcc** 2.95.3qnx-nto 20010315:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

The following binaries are built on third-party systems kindly provided to MySQL AB by other users. These are provided only as a courtesy; MySQL AB does not have full control over these systems, so we can provide only limited support for the binaries built on them.

- SCO Unix 3.2v5.0.7 i386 with **gcc** 2.95.3:

```
CFLAGS="-O3 -mpentium" LDFLAGS=-static CXX=gcc CXXFLAGS="-O3 -mp
-felide-constructors" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-saf
--disable-shared
```

- SCO UnixWare 7.1.4 i386 with **CC** 3.2:

```
CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-saf
--disable-shared --with-readline
```

- SCO OpenServer 6.0.0 i386 with **CC** 3.2:

```
CC=cc CFLAGS="-O" CXX=CC ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --with-named-z-libs=no --enable-thread-saf
--disable-shared --with-readline
```

- Compaq Tru64 OSF/1 V5.1 732 alpha with **cc/cxx** (Compaq C V6.3-029i / DIGITAL C++ V6.1-027):

```
CC="cc -pthread" CFLAGS="-O4 -ansi_alias -ansi_args -fast -inlir
speed -speculate all" CXX="cxx -pthread" CXXFLAGS="-O4 -ansi_ali
```

```
-fast -inline speed -speculate all -noexceptions -nortti" ./conf
--prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile
--with-named-thread-libs="-lpthread -lmach -lexc -lc" --disable-
--with-mysqld-ldflags=-all-static
```

- SGI Irix 6.5 IP32 with **gcc** 3.0.1:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared
```

- FreeBSD/sparc64 5.0 with **gcc** 3.2.1:

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/my
--localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/my
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --disable-shared --with-innodb
```

The following compile options have been used for binary packages that MySQL AB provided in the past. These binaries no longer are being updated, but the compile options are listed here for reference purposes.

- Linux 2.2.xx SPARC with **egcs** 1.1.2:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3
-fno-omit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex --enable-thread-safe-client
--enable-local-infile --enable-assembler --disable-shared
```

- Linux 2.2.x with x686 with **gcc** 2.95.2:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro
-felide-constructors -fno-exceptions -fno-rtti" ./configure
--prefix=/usr/local/mysql --enable-assembler
--with-mysqld-ldflags=-all-static --disable-shared
--with-extra-charsets=complex
```

- SunOS 4.1.4 2 sun4c with **gcc** 2.7.2.1:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors" ./configure
--prefix=/usr/local/mysql --disable-shared --with-extra-charsets
--enable-assembler
```

- SunOS 5.5.1 (and above) sun4u with **egcs** 1.0.3a or 2.90.27 or **gcc** 2.95.2 and newer:

```
CC=gcc CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql
--with-low-memory --with-extra-charsets=complex --enable-assembly
```

- SunOS 5.6 i86pc with **gcc** 2.8.1:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-low-memory --with-extra-charsets=complex
```

- BSDI BSD/OS 3.1 i386 with **gcc** 2.7.2.1:

```
CC=gcc CXX=gcc CXXFLAGS=-O ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex
```

- BSDI BSD/OS 2.1 i386 with **gcc** 2.7.2:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex
```

- AIX 4.2 with **gcc** 2.7.2.2:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex
```

2.1.3. How to Get MySQL

Check our downloads page at <http://dev.mysql.com/downloads/> for information about the current version of MySQL and for downloading instructions. For a complete up-to-date list of MySQL download mirror sites, see <http://dev.mysql.com/downloads/mirrors.html>. You can also find information there about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

Our main mirror is located at <http://mirrors.sunsite.dk/mysql/>.

2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG

After you have downloaded the MySQL package that suits your needs and

before you attempt to install it, you should make sure that it is intact and has not been tampered with. MySQL AB offers three means of integrity checking:

- MD5 checksums
- Cryptographic signatures using GnuPG, the GNU Privacy Guard
- For RPM packages, the built-in RPM integrity verification mechanism

The following sections describe how to use these methods.

If you notice that the MD5 checksum or GPG signatures do not match, first try to download the respective package one more time, perhaps from another mirror site. If you repeatedly cannot successfully verify the integrity of the package, please notify us about such incidents, including the full package name and the download site you have been using, at <webmaster@mysql.com> or <build@mysql.com>. Do not report downloading problems using the bug-reporting system.

2.1.4.1. Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that you can verify with the following command, where *package_name* is the name of the package you downloaded:

```
shell> md5sum package_name
```

Example:

```
shell> md5sum mysql-standard-5.0.25-linux-i686.tar.gz  
aaab65abbec64d5e907dcd41b8699945  mysql-standard-5.0.25-linux-i686.t
```

You should verify that the resulting checksum (the string of hexadecimal digits) matches the one displayed on the download page immediately below the respective package.

Note: Make sure to verify the checksum of the *archive file* (for example, the .zip or .tar.gz file) and not of the files that are contained inside of the archive.

Note that not all operating systems support the **md5sum** command. On some, it is simply called **md5**, and others do not ship it at all. On Linux, it is part of the **GNU Text Utilities** package, which is available for a wide range of platforms. You can download the source code from <http://www.gnu.org/software/textutils/> as well. If you have OpenSSL installed, you can use the command **openssl md5 *package_name*** instead. A Windows implementation of the **md5** command line utility is available from <http://www.fourmilab.ch/md5/>. **winMd5Sum** is a graphical MD5 checking tool that can be obtained from <http://www.nullriver.com/index/products/winmd5sum>.

2.1.4.2. Signature Checking Using GnuPG

Another method of verifying the integrity and authenticity of a package is to use cryptographic signatures. This is more reliable than using MD5 checksums, but requires more work.

At MySQL AB, we sign MySQL downloadable packages with **GnuPG** (GNU Privacy Guard). **GnuPG** is an Open Source alternative to the well-known Pretty Good Privacy (**PGP**) by Phil Zimmermann. See <http://www.gnupg.org/> for more information about **GnuPG** and how to obtain and install it on your system. Most Linux distributions ship with **GnuPG** installed by default. For more information about **GnuPG**, see <http://www.openpgp.org/>.

To verify the signature for a specific package, you first need to obtain a copy of MySQL AB's public GPG build key, which you can download from <http://www.keyserver.net/>. The key that you want to obtain is named `build@mysql.com`. Alternatively, you can cut and paste the key directly from the following text:

```
Key ID:
pub 1024D/5072E1F5 2003-02-03
    MySQL Package signing key (www.mysql.com) <build@mysql.com>
Fingerprint: A4A9 4068 76FC BD3C 4567 70C8 8C71 8D3B 5072 E1F5
```

Public Key (ASCII-armored):

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org
```

```
mQGIBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bP1UWC3
R0DjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBDck96+OmSLN9brZ
```

```
fW2v0UgCmYv2hW0hyDHuvY1QA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
Bq0xRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rF2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQC7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2Zo0HODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNLYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPANWqcyefep
rv0sxxqUEMcM3o7wwgfn83P0kDasDbs3pjwPhxvzh6//62zQJ7Q7TX1TUUwgUGFj
a2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkqPGJ1aWxkQG15c3FsLmNv
bT6IXQQTEQIAHQUCPj6jDAUJcWYBgAULBwoDBAMVAwIDFgIBAheAAAoJEIxxjTtQ
cuH1cY4AnilUwTXn8MatQ0iG0a/bPxrvK/gCAJ4oinSNZRYTnb1ChwFaazt7PF3q
zIhMBMRAGAMBQI+PqPRBYMJZgC7AAoJEElQ4SqycpHyJOEAn1mxHijft00bKXvu
cSo/pECUmppiAJ41M9MRVj5VcdH/KN/KjRtW6tHFPYhMBMRAGAMBQI+QoIDBYMJ
YiKJAAoJELb1zU3GuiQ/lpEaoIhpp6BozKI8p6eaabzF5MlJH58pAKCu/R0ofK8J
Eg2aLos+5zEYrB/LsrkCDQQ+PqMdEAgA7+GJfxbMdY4ws1PnjH9rF4N2qfwsEN/l
xaZoJYc3a6M02WCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb601JHff0o2HfLmRDRi
Rjd1DTCHqeyX7CHhcghj/dNRlW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAWzE
7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWmu0iM1BzELqX1DY7LwoPEb/09Rkbf4fm
Le11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8I1bf5vSYHbuE5p
/1oIDznkg/p8kw+3FxuWrycciqFTcNz215yyX39LXFnlLzKub/F5GwADBQf+Lwqq
a8CGrRfs0AJxim63CHfty5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1ZaSaf
anFvwFG6Llx9xkU7tzq+vKLowkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGoTb0W
I39Ecyzgs1zBdC++MPjcQTcA7p6JUvS6p0AB3FQwg54tuUo0Ec8bsM8b3Ev42Lmu
QT5NdKHGwHsXTptl0klk4bQk40ajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkKht92
6s/ymfdf5HkdQ1cyvsz5tryVI3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdU0etdZ
Whe70YGNPw1yjWJT1IhMBBgRAGAMBQI+PqMdBQkJZgGAAoJEIxxjTtQcuH17p4A
n3r1QpVC9yhnW2cSAjq+kr72GX0eAJ4295k16NxYEuFApmr1+0uUq/SlsQ==
=YJkx
-----END PGP PUBLIC KEY BLOCK-----
```

To import the build key into your personal public GPG keyring, use **gpg --import**. For example, if you have saved the key in a file named `mysql_pubkey.asc`, the import command looks like this:

```
shell> gpg --import mysql_pubkey.asc
```

After you have downloaded and imported the public build key, download your desired MySQL package and the corresponding signature, which also is available from the download page. The signature file has the same name as the distribution file with an `.asc` extension. For example:

Distribution file	mysql-standard-5.0.25-linux-i686.tar.gz
Signature file	mysql-standard-5.0.25-linux-i686.tar.gz.asc

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file:

```
shell> gpg --verify package_name.asc
```

Example:

```
shell> gpg --verify mysql-standard-5.0.25-linux-i686.tar.gz.asc
gpg: Signature made Tue 12 Jul 2005 23:35:41 EST using DSA key ID 50
gpg: Good signature from "MySQL Package signing key (www.mysql.com)
```

The Good signature message indicates that everything is all right. You can ignore any insecure memory warning you might obtain.

See the GPG documentation for more information on how to work with public keys.

2.1.4.3. Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages have a built-in GPG signature and MD5 checksum. You can verify a package by running the following command:

```
shell> rpm --checksig package_name.rpm
```

Example:

```
shell> rpm --checksig MySQL-server-5.0.25-0.i386.rpm
MySQL-server-5.0.25-0.i386.rpm: md5 gpg OK
```

Note: If you are using RPM 4.1 and it complains about (GPG) NOT OK (MISSING KEYS: GPG#5072e1f5), even though you have imported the MySQL public build key into your own GPG keyring, you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (or GPG itself). Rather, it maintains its own keyring because it is a system-wide application and a user's GPG public keyring is a user-specific file. To import the MySQL public key into the RPM keyring, first obtain the key as described in [Section 2.1.4.2, “Signature Checking Using GnuPG”](#). Then use **rpm --import** to import the key. For example, if you have saved the public key in a file named `mysql_pubkey.asc`, import it using this command:

```
shell> rpm --import mysql_pubkey.asc
```

If you need to obtain the MySQL public key, see [Section 2.1.4.2, “Signature](#)

[Checking Using GnuPG](#)".

2.1.5. Installation Layouts

This section describes the default layout of the directories created by installing binary or source distributions provided by MySQL AB. A distribution provided by another vendor might use a layout different from those shown here.

For MySQL 5.0 on Windows, the default installation directory is C:\Program Files\MySQL\MySQL Server 5.0. (Some Windows users prefer to install in C:\mysql, the directory that formerly was used as the default. However, the layout of the subdirectories remains the same.) The installation directory has the following subdirectories:

Directory	Contents of Directory
bin	Client programs and the mysqld server
data	Log files, databases
Docs	Documentation
examples	Example programs and scripts
include	Include (header) files
lib	Libraries
scripts	Utility scripts
share	Error message files

Installations created from MySQL AB's Linux RPM distributions result in files under the following system directories:

Directory	Contents of Directory
/usr/bin	Client programs and scripts
/usr/sbin	The mysqld server
/var/lib/mysql	Log files, databases
/usr/share/doc/packages	Documentation
/usr/include/mysql	Include (header) files
/usr/lib/mysql	Libraries
/usr/share/mysql	

	Error message and character set files
/usr/share/sql-bench	Benchmarks

On Unix, a **tar** file binary distribution is installed by unpacking it at the installation location you choose (typically /usr/local/mysql) and creates the following directories in that location:

Directory	Contents of Directory
bin	Client programs and the mysqld server
data	Log files, databases
docs	Documentation, ChangeLog
include	Include (header) files
lib	Libraries
scripts	mysql_install_db
share/mysql	Error message files
sql-bench	Benchmarks

A source distribution is installed after you configure and compile it. By default, the installation step installs files under /usr/local, in the following subdirectories:

Directory	Contents of Directory
bin	Client programs and scripts
include/mysql	Include (header) files
info	Documentation in Info format
lib/mysql	Libraries
libexec	The mysqld server
share/mysql	Error message files
sql-bench	Benchmarks and crash-me test
var	Databases and log files

Within its installation directory, the layout of a source installation differs from that of a binary installation in the following ways:

- The **mysqld** server is installed in the `libexec` directory rather than in the `bin` directory.
- The data directory is `var` rather than `data`.
- **mysql_install_db** is installed in the `bin` directory rather than in the `scripts` directory.
- The header file and library directories are `include/mysql` and `lib/mysql` rather than `include` and `lib`.

You can create your own binary installation from a compiled source distribution by executing the `scripts/make_binary_distribution` script from the top directory of the source distribution.

2.2. Standard MySQL Installation Using a Binary Distribution

The next several sections cover the installation of MySQL on platforms where we offer packages using the native packaging format of the respective platform. (This is also known as performing a “binary install.”) However, binary distributions of MySQL are available for many other platforms as well. See [Section 2.8, “Installing MySQL on Other Unix-Like Systems”](#), for generic installation instructions for these packages that apply to all platforms.

See [Section 2.1, “General Installation Issues”](#), for more information on what other binary distributions are available and how to obtain them.

2.3. Installing MySQL on Windows

A native Windows distribution of MySQL has been available from MySQL AB since version 3.21 and represents a sizable percentage of the daily downloads of MySQL. This section describes the process for installing MySQL on Windows.

Note: If you are upgrading MySQL from an existing installation older than MySQL 4.1.5, you must first perform the procedure described in [Section 2.3.14, “Upgrading MySQL on Windows”](#).

To run MySQL on Windows, you need the following:

- A 32-bit Windows operating system such as 9x, Me, NT, 2000, XP, or Windows Server 2003.

A Windows NT-based operating system (NT, 2000, XP, 2003) permits you to run the MySQL server as a service. The use of a Windows NT-based operating system is strongly recommended. See [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

Generally, you should install MySQL on Windows using an account that has administrator rights. Otherwise, you may encounter problems with certain operations such as editing the PATH environment variable or accessing the **Service Control Manager**.

- TCP/IP protocol support.
- Enough space on the hard drive to unpack, install, and create the databases in accordance with your requirements (generally a minimum of 200 megabytes is recommended.)

There may also be other requirements, depending on how you plan to use MySQL:

- If you plan to connect to the MySQL server via ODBC, you need a Connector/ODBC driver. See [Chapter 23, Connectors](#).
- If you need tables with a size larger than 4GB, install MySQL on an NTFS

or newer filesystem. Don't forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See [Section 13.1.5, “CREATE TABLE Syntax”](#).

MySQL for Windows is available in several distribution formats:

- Binary distributions are available that contain a setup program that installs everything you need so that you can start the server immediately. Another binary distribution format contains an archive that you simply unpack in the installation location and then configure yourself. For details, see [Section 2.3.1, “Choosing An Installation Package”](#).
- The source distribution contains all the code and support files for building the executables using the Visual Studio 7.1 compiler system.

Generally speaking, you should use a binary distribution that includes an installer. It is simpler to use than the others, and you need no additional tools to get MySQL up and running. The installer for the Windows version of MySQL, combined with a GUI Configuration Wizard, automatically installs MySQL, creates an option file, starts the server, and secures the default user accounts.

The following section describes how to install MySQL on Windows using a binary distribution. To use an installation package that does not include an installer, follow the procedure described in [Section 2.3.5, “Installing MySQL from a Noinstall Zip Archive”](#). To install using a source distribution, see [Section 2.9.6, “Installing MySQL from Source on Windows”](#).

MySQL distributions for Windows can be downloaded from <http://dev.mysql.com/downloads/>. See [Section 2.1.3, “How to Get MySQL”](#).

2.3.1. Choosing An Installation Package

For MySQL 5.0, there are three installation packages to choose from when installing MySQL on Windows:

- **The Essentials Package:** This package has a filename similar to `mysql-essential-5.0.25-win32.msi` and contains the minimum set of files needed to install MySQL on Windows, including the Configuration Wizard. This package does not include optional components such as the embedded server and benchmark suite.

- **The Complete Package:** This package has a filename similar to `mysql-5.0.25-win32.zip` and contains all files needed for a complete Windows installation, including the Configuration Wizard. This package includes optional components such as the embedded server and benchmark suite.
- **The Noinstall Archive:** This package has a filename similar to `mysql-noinstall-5.0.25-win32.zip` and contains all the files found in the Complete install package, with the exception of the Configuration Wizard. This package does not include an automated installer, and must be manually installed and configured.

The Essentials package is recommended for most users. It is provided as an `.msi` file for use with the Windows Installer. The Complete and Noinstall distributions are packaged as Zip archives. To use them, you must have a tool that can unpack `.zip` files.

Your choice of install package affects the installation process you must follow. If you choose to install either the Essentials or Complete install packages, see [Section 2.3.2, “Installing MySQL with the Automated Installer”](#). If you choose to install MySQL from the Noinstall archive, see [Section 2.3.5, “Installing MySQL from a Noinstall Zip Archive”](#).

2.3.2. Installing MySQL with the Automated Installer

New MySQL users can use the MySQL Installation Wizard and MySQL Configuration Wizard to install MySQL on Windows. These are designed to install and configure MySQL in such a way that new users can immediately get started using MySQL.

The MySQL Installation Wizard and MySQL Configuration Wizard are available in the Essentials and Complete install packages. They are recommended for most standard MySQL installations. Exceptions include users who need to install multiple instances of MySQL on a single server host and advanced users who want complete control of server configuration.

2.3.3. Using the MySQL Installation Wizard

2.3.3.1. Introduction to the Installation Wizard

MySQL Installation Wizard is an installer for the MySQL server that uses the latest installer technologies for Microsoft Windows. The MySQL Installation Wizard, in combination with the MySQL Configuration Wizard, allows a user to install and configure a MySQL server that is ready for use immediately after installation.

The MySQL Installation Wizard is the standard installer for all MySQL server distributions, version 4.1.5 and higher. Users of previous versions of MySQL need to shut down and remove their existing MySQL installations manually before installing MySQL with the MySQL Installation Wizard. See [Section 2.3.3.7, “Upgrading MySQL with the Installation Wizard”](#), for more information on upgrading from a previous version.

Microsoft has included an improved version of their Microsoft Windows Installer (MSI) in the recent versions of Windows. MSI has become the de-facto standard for application installations on Windows 2000, Windows XP, and Windows Server 2003. The MySQL Installation Wizard makes use of this technology to provide a smoother and more flexible installation process.

The Microsoft Windows Installer Engine was updated with the release of Windows XP; those using a previous version of Windows can reference [this Microsoft Knowledge Base article](#) for information on upgrading to the latest version of the Windows Installer Engine.

In addition, Microsoft has introduced the WiX (Windows Installer XML) toolkit recently. This is the first highly acknowledged Open Source project from Microsoft. We have switched to WiX because it is an Open Source project and it allows us to handle the complete Windows installation process in a flexible manner using scripts.

Improving the MySQL Installation Wizard depends on the support and feedback of users like you. If you find that the MySQL Installation Wizard is lacking some feature important to you, or if you discover a bug, please report it in our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#).

2.3.3.2. Downloading and Starting the MySQL Installation Wizard

The MySQL installation packages can be downloaded from

<http://dev.mysql.com/downloads/>. If the package you download is contained within a Zip archive, you need to extract the archive first.

The process for starting the wizard depends on the contents of the installation package you download. If there is a `setup.exe` file present, double-click it to start the installation process. If there is an `.msi` file present, double-click it to start the installation process.

2.3.3.3. Choosing an Install Type

There are three installation types available: **Typical**, **Complete**, and **Custom**.

The **Typical** installation type installs the MySQL server, the **mysql** command-line client, and the command-line utilities. The command-line clients and utilities include **mysqldump**, **myisamchk**, and several other tools to help you manage the MySQL server.

The **Complete** installation type installs all components included in the installation package. The full installation package includes components such as the embedded server library, the benchmark suite, support scripts, and documentation.

The **Custom** installation type gives you complete control over which packages you wish to install and the installation path that is used. See [Section 2.3.3.4, “The Custom Install Dialog”](#), for more information on performing a custom install.

If you choose the **Typical** or **Complete** installation types and click the Next button, you advance to the confirmation screen to verify your choices and begin the installation. If you choose the **Custom** installation type and click the Next button, you advance to the custom installation dialog, described in [Section 2.3.3.4, “The Custom Install Dialog”](#).

2.3.3.4. The Custom Install Dialog

If you wish to change the installation path or the specific components that are installed by the MySQL Installation Wizard, choose the **Custom** installation type.

A tree view on the left side of the custom install dialog lists all available components. Components that are not installed have a red X icon; components that are installed have a gray icon. To change whether a component is installed, click on that component's icon and choose a new option from the drop-down list that appears.

You can change the default installation path by clicking the Change... button to the right of the displayed installation path.

After choosing your installation components and installation path, click the Next button to advance to the confirmation dialog.

2.3.3.5. The Confirmation Dialog

Once you choose an installation type and optionally choose your installation components, you advance to the confirmation dialog. Your installation type and installation path are displayed for you to review.

To install MySQL if you are satisfied with your settings, click the Install button. To change your settings, click the Back button. To exit the MySQL Installation Wizard without installing MySQL, click the Cancel button.

After installation is complete, you have the option of registering with the MySQL web site. Registration gives you access to post in the MySQL forums at forums.mysql.com, along with the ability to report bugs at bugs.mysql.com and to subscribe to our newsletter. The final screen of the installer provides a summary of the installation and gives you the option to launch the MySQL Configuration Wizard, which you can use to create a configuration file, install the MySQL service, and configure security settings.

2.3.3.6. Changes Made by MySQL Installation Wizard

Once you click the Install button, the MySQL Installation Wizard begins the installation process and makes certain changes to your system which are described in the sections that follow.

Changes to the Registry

The MySQL Installation Wizard creates one Windows registry key in a typical

install situation, located in HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB.

The MySQL Installation Wizard creates a key named after the major version of the server that is being installed, such as MySQL Server 5.0. It contains two string values, Location and Version. The Location string contains the path to the installation directory. In a default installation it contains C:\Program Files\MySQL\MySQL Server 5.0\.. The Version string contains the release number. For example, for an installation of MySQL Server 5.0.25, the key contains a value of 5.0.25.

These registry keys are used to help external tools identify the installed location of the MySQL server, preventing a complete scan of the hard-disk to determine the installation path of the MySQL server. The registry keys are not required to run the server, and if you install MySQL using the noinstall Zip archive, the registry keys are not created.

Changes to the Start Menu

The MySQL Installation Wizard creates a new entry in the Windows Start menu under a common MySQL menu heading named after the major version of MySQL that you have installed. For example, if you install MySQL 5.0, the MySQL Installation Wizard creates a MySQL Server 5.0 section in the Start menu.

The following entries are created within the new Start menu section:

- **MySQL Command Line Client:** This is a shortcut to the **mysql** command-line client and is configured to connect as the root user. The shortcut prompts for a root user password when you connect.
- **MySQL Server Instance Config Wizard:** This is a shortcut to the MySQL Configuration Wizard. Use this shortcut to configure a newly installed server, or to reconfigure an existing server.
- **MySQL Documentation:** This is a link to the MySQL server documentation that is stored locally in the MySQL server installation directory. This option is not available when the MySQL server is installed using the Essentials installation package.

Changes to the File System

The MySQL Installation Wizard by default installs the MySQL 5.0 server to `C:\Program Files\MySQL\MySQL Server 5.0`, where *Program Files* is the default location for applications in your system, and *5.0* is the major version of your MySQL server. This is the recommended location for the MySQL server, replacing the former default location `C:\mysql`.

By default, all MySQL applications are stored in a common directory at `C:\Program Files\MySQL`, where *Program Files* is the default location for applications in your Windows installation. A typical MySQL installation on a developer machine might look like this:

```
C:\Program Files\MySQL\MySQL Server 5.0
C:\Program Files\MySQL\MySQL Administrator 1.0
C:\Program Files\MySQL\MySQL Query Browser 1.0
```

This approach makes it easier to manage and maintain all MySQL applications installed on a particular system.

2.3.3.7. Upgrading MySQL with the Installation Wizard

The MySQL Installation Wizard can perform server upgrades automatically using the upgrade capabilities of MSI. That means you do not need to remove a previous installation manually before installing a new release. The installer automatically shuts down and removes the previous MySQL service before installing the new version.

Automatic upgrades are available only when upgrading between installations that have the same major and minor version numbers. For example, you can upgrade automatically from MySQL 4.1.5 to MySQL 4.1.6, but not from MySQL 4.1 to MySQL 5.0.

See [Section 2.3.14, “Upgrading MySQL on Windows”](#).

2.3.4. Using the Configuration Wizard

2.3.4.1. Introduction to the Configuration Wizard

The MySQL Configuration Wizard helps automate the process of configuring your server under Windows. The MySQL Configuration Wizard creates a

custom `my.ini` file by asking you a series of questions and then applying your responses to a template to generate a `my.ini` file that is tuned to your installation.

The MySQL Configuration Wizard is included with the MySQL 5.0 server, and is currently available for Windows users only.

The MySQL Configuration Wizard is to a large extent the result of feedback that MySQL AB has received from many users over a period of several years. However, if you find that it lacks some feature important to you, please report it in our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#).

2.3.4.2. Starting the MySQL Configuration Wizard

The MySQL Configuration Wizard is typically launched from the MySQL Installation Wizard, as the MySQL Installation Wizard exits. You can also launch the MySQL Configuration Wizard by clicking the MySQL Server Instance Config Wizard entry in the MySQL section of the Windows Start menu.

Alternatively, you can navigate to the `bin` directory of your MySQL installation and launch the `MySQLInstanceConfig.exe` file directly.

2.3.4.3. Choosing a Maintenance Option

If the MySQL Configuration Wizard detects an existing `my.ini` file, you have the option of either reconfiguring your existing server, or removing the server instance by deleting the `my.ini` file and stopping and removing the MySQL service.

To reconfigure an existing server, choose the Re-configure Instance option and click the Next button. Your existing `my.ini` file is renamed to `mytimestamp.ini.bak`, where *timestamp* is the date and time at which the existing `my.ini` file was created. To remove the existing server instance, choose the Remove Instance option and click the Next button.

If you choose the Remove Instance option, you advance to a confirmation window. Click the Execute button. The MySQL Configuration Wizard stops and removes the MySQL service, and then deletes the `my.ini` file. The server

installation and its data folder are not removed.

If you choose the Re-configure Instance option, you advance to the Configuration Type dialog where you can choose the type of installation that you wish to configure.

2.3.4.4. Choosing a Configuration Type

When you start the MySQL Configuration Wizard for a new MySQL installation, or choose the Re-configure Instance option for an existing installation, you advance to the Configuration Type dialog.

There are two configuration types available: Detailed Configuration and Standard Configuration. The Standard Configuration option is intended for new users who want to get started with MySQL quickly without having to make many decisions about server configuration. The Detailed Configuration option is intended for advanced users who want more fine-grained control over server configuration.

If you are new to MySQL and need a server configured as a single-user developer machine, the Standard Configuration should suit your needs. Choosing the Standard Configuration option causes the MySQL Configuration Wizard to set all configuration options automatically with the exception of Service Options and Security Options.

The Standard Configuration sets options that may be incompatible with systems where there are existing MySQL installations. If you have an existing MySQL installation on your system in addition to the installation you wish to configure, the Detailed Configuration option is recommended.

To complete the Standard Configuration, please refer to the sections on Service Options and Security Options in [Section 2.3.4.11, “The Service Options Dialog”](#), and [Section 2.3.4.12, “The Security Options Dialog”](#), respectively.

2.3.4.5. The Server Type Dialog

There are three different server types available to choose from. The server type that you choose affects the decisions that the MySQL Configuration Wizard makes with regard to memory, disk, and processor usage.

- **Developer Machine:** Choose this option for a typical desktop workstation where MySQL is intended only for personal use. It is assumed that many other desktop applications are running. The MySQL server is configured to use minimal system resources.
- **Server Machine:** Choose this option for a server machine where the MySQL server is running alongside other server applications such as FTP, email, and Web servers. The MySQL server is configured to use a moderate portion of the system resources.
- **Dedicated MySQL Server Machine:** Choose this option for a server machine that is intended to run only the MySQL server. It is assumed that no other applications are running. The MySQL server is configured to use all available system resources.

2.3.4.6. The Database Usage Dialog

The Database Usage dialog allows you to indicate the storage engines that you expect to use when creating MySQL tables. The option you choose determines whether the InnoDB storage engine is available and what percentage of the server resources are available to InnoDB.

- **Multifunctional Database:** This option enables both the InnoDB and MyISAM storage engines and divides resources evenly between the two. This option is recommended for users who use both storage engines on a regular basis.
- **Transactional Database Only:** This option enables both the InnoDB and MyISAM storage engines, but dedicates most server resources to the InnoDB storage engine. This option is recommended for users who use InnoDB almost exclusively and make only minimal use of MyISAM.
- **Non-Transactional Database Only:** This option disables the InnoDB storage engine completely and dedicates all server resources to the MyISAM storage engine. This option is recommended for users who do not use InnoDB.

2.3.4.7. The InnoDB Tablespace Dialog

Some users may want to locate the InnoDB tablespace files in a different location than the MySQL server data directory. Placing the tablespace files in a separate

location can be desirable if your system has a higher capacity or higher performance storage device available, such as a RAID storage system.

To change the default location for the InnoDB tablespace files, choose a new drive from the drop-down list of drive letters and choose a new path from the drop-down list of paths. To create a custom path, click the ... button.

If you are modifying the configuration of an existing server, you must click the Modify button before you change the path. In this situation you must move the existing tablespace files to the new location manually before starting the server.

2.3.4.8. The Concurrent Connections Dialog

To prevent the server from running out of resources, it is important to limit the number of concurrent connections to the MySQL server that can be established. The Concurrent Connections dialog allows you to choose the expected usage of your server, and sets the limit for concurrent connections accordingly. It is also possible to set the concurrent connection limit manually.

- **Decision Support (DSS)/OLAP:** Choose this option if your server does not require a large number of concurrent connections. The maximum number of connections is set at 100, with an average of 20 concurrent connections assumed.
- **Online Transaction Processing (OLTP):** Choose this option if your server requires a large number of concurrent connections. The maximum number of connections is set at 500.
- **Manual Setting:** Choose this option to set the maximum number of concurrent connections to the server manually. Choose the number of concurrent connections from the drop-down box provided, or enter the maximum number of connections into the drop-down box if the number you desire is not listed.

2.3.4.9. The Networking and Strict Mode Options Dialog

Use the Networking Options dialog to enable or disable TCP/IP networking and to configure the port number that is used to connect to the MySQL server.

TCP/IP networking is enabled by default. To disable TCP/IP networking, uncheck the box next to the Enable TCP/IP Networking option.

Port 3306 is used by default. To change the port used to access MySQL, choose a new port number from the drop-down box or type a new port number directly into the drop-down box. If the port number you choose is in use, you are prompted to confirm your choice of port number.

Set the Server SQL Mode to either enable or disable strict mode. Enabling strict mode (default) makes MySQL behave more like other database management systems. *If you run applications that rely on MySQL's old “forgiving” behavior, make sure to either adapt those applications or to disable strict mode.* For more information about strict mode, see [Section 5.2.5, “The Server SQL Mode”](#).

2.3.4.10. The Character Set Dialog

The MySQL server supports multiple character sets and it is possible to set a default server character set that is applied to all tables, columns, and databases unless overridden. Use the Character Set dialog to change the default character set of the MySQL server.

- **Standard Character Set:** Choose this option if you want to use `latin1` as the default server character set. `latin1` is used for English and many Western European languages.
- **Best Support For Multilingualism:** Choose this option if you want to use `utf8` as the default server character set. This is a Unicode character set that can store characters from many different languages.
- **Manual Selected Default Character Set / Collation:** Choose this option if you want to pick the server's default character set manually. Choose the desired character set from the provided drop-down list.

2.3.4.11. The Service Options Dialog

On Windows NT-based platforms, the MySQL server can be installed as a Windows service. When installed this way, the MySQL server can be started automatically during system startup, and even restarted automatically by Windows in the event of a service failure.

The MySQL Configuration Wizard installs the MySQL server as a service by default, using the service name `MySQL`. If you do not wish to install the service, uncheck the box next to the Install As Windows Service option. You can change the service name by picking a new service name from the drop-down box provided or by entering a new service name into the drop-down box.

To install the MySQL server as a service but not have it started automatically at startup, uncheck the box next to the Launch the MySQL Server Automatically option.

2.3.4.12. The Security Options Dialog

It is strongly recommended that you set a root password for your MySQL server, and the MySQL Configuration Wizard requires by default that you do so. If you do not wish to set a root password, uncheck the box next to the Modify Security Settings option.

To set the root password, enter the desired password into both the New root password and Confirm boxes. If you are reconfiguring an existing server, you need to enter the existing root password into the Current root password box.

To prevent root logins from across the network, check the box next to the Root may only connect from localhost option. This increases the security of your root account.

To create an anonymous user account, check the box next to the Create An Anonymous Account option. Creating an anonymous account can decrease server security and cause login and permission difficulties. For this reason, it is not recommended.

2.3.4.13. The Confirmation Dialog

The final dialog in the MySQL Configuration Wizard is the Confirmation Dialog. To start the configuration process, click the Execute button. To return to a previous dialog, click the Back button. To exit the MySQL Configuration Wizard without configuring the server, click the Cancel button.

After you click the Execute button, the MySQL Configuration Wizard performs a series of tasks and displays the progress onscreen as the tasks are performed.

The MySQL Configuration Wizard first determines configuration file options based on your choices using a template prepared by MySQL AB developers and engineers. This template is named `my-template.ini` and is located in your server installation directory.

The MySQL Configuration Wizard then writes these options to a `my.ini` file. The final location of the `my.ini` file is displayed next to the Write configuration file task.

If you chose to create a service for the MySQL server, the MySQL Configuration Wizard creates and starts the service. If you are reconfiguring an existing service, the MySQL Configuration Wizard restarts the service to apply your configuration changes.

If you chose to set a root password, the MySQL Configuration Wizard connects to the server, sets your new root password and applies any other security settings you may have selected.

After the MySQL Configuration Wizard has completed its tasks, it displays a summary. Click the Finish button to exit the MySQL Configuration Wizard.

2.3.4.14. The Location of the my.ini File

The MySQL Configuration Wizard places the `my.ini` file in the installation directory for the MySQL server. This helps associate configuration files with particular server instances.

To ensure that the MySQL server knows where to look for the `my.ini` file, an argument similar to this is passed to the MySQL server as part of the service installation:

```
--defaults-file="C:\Program Files\MySQL\MySQL Server 5.0\my.ini"
```

Here, `C:\Program Files\MySQL\MySQL Server 5.0` is replaced with the installation path to the MySQL Server. The `--defaults-file` option instructs the MySQL server to read the specified file for configuration options when it starts.

2.3.4.15. Editing the my.ini File

To modify the `my.ini` file, open it with a text editor and make any necessary changes. You can also modify the server configuration with the [MySQL Administrator](#) utility.

MySQL clients and utilities such as the **mysql** and **mysqldump** command-line clients are not able to locate the `my.ini` file located in the server installation directory. To configure the client and utility applications, create a new `my.ini` file in the `C:\WINDOWS` or `C:\WINNT` directory (whichever is applicable to your Windows version).

2.3.5. Installing MySQL from a Noinstall Zip Archive

Users who are installing from the Noinstall package can use the instructions in this section to manually install MySQL. The process for installing MySQL from a Zip archive is as follows:

1. Extract the archive to the desired install directory
2. Create an option file
3. Choose a MySQL server type
4. Start the MySQL server
5. Secure the default user accounts

This process is described in the sections that follow.

2.3.6. Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to [Section 2.3.14, “Upgrading MySQL on Windows”](#), before beginning the upgrade process.
2. If you are using a Windows NT-based operating system such as Windows NT, Windows 2000, Windows XP, or Windows Server 2003, make sure that you are logged in as a user with administrator privileges.
3. Choose an installation location. Traditionally, the MySQL server is installed

in `C:\mysql`. The MySQL Installation Wizard installs MySQL under `C:\Program Files\MySQL`. If you do not install MySQL at `C:\mysql`, you must specify the path to the install directory during startup or in an option file. See [Section 2.3.7, “Creating an Option File”](#).

4. Extract the install archive to the chosen installation location using your preferred Zip archive tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs, you can move the contents of the subfolder into the chosen installation location.

2.3.7. Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations are different from the default locations (`C:\Program Files\MySQL\MySQL Server 5.0` and `C:\Program Files\MySQL\MySQL Server 5.0\data`).
- You need to tune the server settings.

When the MySQL server starts on Windows, it looks for options in two files: the `my.ini` file in the Windows directory, and the `C:\my.cnf` file. The Windows directory typically is named something like `C:\WINDOWS` or `C:\WINNT`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

MySQL looks for options first in the `my.ini` file, and then in the `my.cnf` file. However, to avoid confusion, it's best if you use only one file. If your PC uses a boot loader where `C:` is not the boot drive, your only option is to use the `my.ini` file. Whichever option file you use, it must be a plain text file.

You can also make use of the example option files included with your MySQL distribution; see [Section 4.3.2.1, “Preconfigured Option Files”](#).

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in E:\mysql and the data directory is in E:\mydata\data, you can create an option file containing a [mysqld] section to specify values for the basedir and datadir parameters:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Note that Windows pathnames are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, you must double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

On Windows, the MySQL installer places the data directory directly under the directory where you install MySQL. If you would like to use a data directory in a different location, you should copy the entire contents of the data directory to the new location. For example, if MySQL is installed in C:\Program Files\MySQL\MySQL Server 5.0, the data directory is by default in C:\Program Files\MySQL\MySQL Server 5.0\data. If you want to use E:\mydata as the data directory instead, you must do two things:

1. Move the entire data directory and all of its contents from C:\Program Files\MySQL\MySQL Server 5.0\data to E:\mydata.
2. Use a --datadir option to specify the new data directory location each time you start the server.

2.3.8. Selecting a MySQL Server type

The following table shows the available servers for Windows in MySQL 5.0:

Binary	Description
mysqld-	Compiled with full debugging and automatic memory allocation

debug	checking, as well as InnoDB and BDB support.
mysqld	Optimized binary with InnoDB support.
mysqld-nt	Optimized binary for Windows NT, 2000, and XP with support for named pipes.
mysqld-max	Optimized binary with InnoDB and BDB support.
mysqld-max-nt	Like mysqld-max , but compiled with support for named pipes.

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

All Windows MySQL 5.0 servers have support for symbolic linking of database directories.

MySQL supports TCP/IP on all Windows platforms. The **mysqld-nt** and **mysqld-max-nt** servers support named pipes on Windows NT, 2000, XP, and 2003. However, the default is to use TCP/IP regardless of platform. (Named pipes are slower than TCP/IP in many Windows configurations.)

Use of named pipes is subject to these conditions:

- Named pipes are enabled only if you start the server with the `--enable-named-pipe` option. It is necessary to use this option explicitly because some users have experienced problems with shutting down the MySQL server when named pipes were used.
- Named-pipe connections are allowed only by the **mysqld-nt** or **mysqld-max-nt** servers, and only if the server is run on a version of Windows that supports named pipes (NT, 2000, XP, 2003).
- These servers can be run on Windows 98 or Me, but only if TCP/IP is installed; named-pipe connections cannot be used.
- These servers cannot be run on Windows 95.

Note: Most of the examples in this manual use **mysqld** as the server name. If you choose to use a different server, such as **mysqld-nt**, make the appropriate substitutions in the commands that are shown in the examples.

2.3.9. Starting the Server for the First Time

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The information here applies primarily if you installed MySQL using the `Noinstall` version, or if you wish to configure and test MySQL manually rather than with the GUI tools.

The examples in these sections assume that MySQL is installed under the default location of `C:\Program Files\MySQL\MySQL Server 5.0`. Adjust the pathnames shown in the examples if you have MySQL installed in a different location.

On NT-based systems such as Windows NT, 2000, XP, or 2003, clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named-pipe connections. For MySQL to work with TCP/IP on Windows NT 4, you must install service pack 3 (or newer).

On Windows 95, 98, or Me, MySQL clients always connect to the server using TCP/IP. (This allows any machine on your network to connect to your MySQL server.) Because of this, you must make sure that TCP/IP support is installed on your machine before starting MySQL. You can find TCP/IP on your Windows CD-ROM.

Note that if you are using an old Windows 95 release (for example, OSR2), it is likely that you have an old Winsock package; MySQL requires Winsock 2. You can get the newest Winsock from <http://www.microsoft.com/>. Windows 98 has the new Winsock 2 library, so it is unnecessary to update the library.

MySQL for Windows also supports shared-memory connections if the server is started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=memory` option.

For information about which server binary to run, see [Section 2.3.8, “Selecting a MySQL Server type”](#).

Testing is best done from a command prompt in a console window (or “DOS window”). In this way you can have the server display status messages in the window where they are easy to see. If something is wrong with your

configuration, these messages make it easier for you to identify and fix any problems.

To start the server, enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --console
```

For a server that includes InnoDB support, you should see the messages similar to those following as it starts (the pathnames and sizes may differ):

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be crea
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be crea
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be crea
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25 InnoDB: Started
```

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
Version: '5.0.25' socket: '' port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the `--console` option, the server writes diagnostic output to the error log in the data directory (C:\Program Files\MySQL\MySQL Server 5.0\data by default). The error log is the file with the `.err` extension.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.3.10. Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

To start the **mysqld** server from the command line, you should start a console window (or “DOS window”) and enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld"
```

The path to **mysqld** may vary depending on the install location of MySQL on your system.

On non-NT versions of Windows, this command starts **mysqld** in the background. That is, after the server starts, you should see another command prompt. If you start the server this way on Windows NT, 2000, XP, or 2003, the server runs in the foreground and no command prompt appears until the server exits. Because of this, you should open another console window to run client programs while the server is running.

You can stop the MySQL server by executing this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin" -u root
```

Note: If the MySQL root user account has a password, you need to invoke **mysqladmin** with the **-p** option and supply the password when prompted.

This command invokes the MySQL administrative utility **mysqladmin** to connect to the server and tell it to shut down. The command connects as the MySQL root user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

If **mysqld** doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the C:\Program Files\MySQL\MySQL Server 5.0\data directory. It is the file with a suffix of **.err**. You can also try to start the server as **mysqld --console**; in this case, you may get some useful information on the screen that may help solve the problem.

The last option is to start **mysqld** with the **--standalone** and **--debug** options. In this case, **mysqld** writes a log file C:\mysqld.trace that should contain the

reason why **mysqld** doesn't start. See [Section E.1.2, "Creating Trace Files"](#).

Use **mysqld --verbose --help** to display all the options that **mysqld** understands.

2.3.11. Starting MySQL as a Windows Service

On the NT family (Windows NT, 2000, XP, 2003), the recommended way to run MySQL is to install it as a Windows service, whereby MySQL starts and stops automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using **NET** commands, or with the graphical **Services** utility.

The **Services** utility (the Windows **Service Control Manager**) can be found in the Windows Control Panel (under Administrative Tools on Windows 2000, XP, and Server 2003). To avoid conflicts, it is advisable to close the **Services** utility while performing server installation or removal operations from the command line.

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin" -u root
```

Note: If the MySQL root user account has a password, you need to invoke **mysqladmin** with the **-p** option and supply the password when prompted.

This command invokes the MySQL administrative utility **mysqladmin** to connect to the server and tell it to shut down. The command connects as the MySQL root user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

Install the server as a service using this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

To make it easier to invoke MySQL programs, you can add the pathname of the

MySQL bin directory to your Windows system PATH environment variable:

- On the Windows desktop, right-click on the My Computer icon, and select Properties
- Next select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.
- Under System Variables, select Path, and then click the Edit button. The Edit System Variable dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked Variable Value. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete pathname of your MySQL bin directory (for example, C:\Program Files\MySQL\MySQL Server 5.0\bin), Note that there should be a semicolon separating this path from any values present in this field. Dismiss this dialogue, and each dialogue in turn, by clicking OK until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the **mysql** client, and all MySQL command-line utilities such as **mysqladmin** and **mysqldump**.

You should not add the MySQL bin directory to your Windows PATH if you are running multiple MySQL servers on the same machine.

Warning: You must exercise great care when editing your system PATH by hand; accidental deletion or modification of any portion of the existing PATH value can leave you with a malfunctioning or even unusable system.

The following additional arguments can be used in MySQL 5.0 when installing the service:

- You can specify a service name immediately following the `--install` option. The default service name is MySQL.
- If a service name is given, it can be followed by a single option. By convention, this should be `--defaults-file=file_name` to specify the name of an option file from which the server should read options when it

starts.

It is possible to use a single option other than `--defaults-file`, but this is discouraged. `--defaults-file` is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file. Also, in MySQL 5.0, use of an option different from `--defaults-file` is not supported until 5.0.3.

- As of MySQL 5.0.1, you can also specify a `--local-service` option following the service name. This causes the server to run using the `LocalService` Windows account that has limited system privileges. This account is available only for Windows XP or newer. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (`MySQL`) following the `--install` option, the server uses the a service name of `MySQL` and reads options from the `[mysqld]` group in the standard option files.
- If the service-installation command specifies a service name other than `MySQL` following the `--install` option, the server uses that service name. It reads options from the group that has the same name as the service, and reads options from the standard option files.

The server also reads options from the `[mysqld]` group from the standard option files. This allows you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group with the same name as a service for use by the server installed with that service name.

- If the service-installation command specifies a `--defaults-file` option after the service name, the server reads options only from the `[mysqld]` group of the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld"
```

```
--install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (MySQL) is given after the `--install` option. If no `--defaults-file` option had been given, this command would have the effect of causing the server to read the `[mysqld]` group from the standard option files. However, because the `--defaults-file` option is present, the server reads options from the `[mysqld]` option group, and only from the named file.

You can also specify options as Start parameters in the Windows **Services** utility before you start the MySQL service.

Once a MySQL server has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from the **Services** utility, or by using a **NET START MySQL** command. The **NET** command is not case sensitive.

When run as a service, **mysqld** has no access to a console window, so no messages can be seen there. If **mysqld** does not start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the MySQL data directory (for example, `C:\Program Files\MySQL\MySQL Server 5.0\data`). It is the file with a suffix of `.err`.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually by using the **Services** utility, the **NET STOP MySQL** command, or the **mysqladmin shutdown** command.

You also have the choice of installing the server as a manual service if you do not wish for the service to be started automatically during the boot process. To do this, use the `--install-manual` option rather than the `--install` option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --install-
```

To remove a server that is installed as a service, first stop it if it is running by executing **NET STOP MySQL**. Then use the `--remove` option to remove it:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --remove
```

If **mysqld** is not running as a service, you can start it from the command line. For instructions, see [Section 2.3.10, “Starting MySQL from the Windows](#)

[Command Line](#)".

Please see [Section 2.3.13, "Troubleshooting a MySQL Installation Under Windows"](#), if you encounter difficulties during installation.

2.3.12. Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlshow"  
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlshow" -u root  
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin" versio  
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysql" test
```

If **mysqld** is slow to respond to TCP/IP connections from client programs, there is probably a problem with your DNS. In this case, start **mysqld** with the `--skip-name-resolve` option and use only localhost and IP numbers in the Host column of the MySQL grant tables.

You can force a MySQL client to use a named-pipe connection rather than TCP/IP by specifying the `--pipe` or `--protocol=PIPE` option, or by specifying `.` (period) as the host name. Use the `--socket` option to specify the name of the pipe if you do not want to use the default pipe name.

Note that if you have set a password for the root account, deleted the anonymous account, or created a new user account, then you must use the appropriate `-u` and `-p` options with the commands shown above in order to connect with the MySQL Server. See [Section 5.8.4, "Connecting to the MySQL Server"](#).

For more information about **mysqlshow**, see [Section 8.15, "mysqlshow — Display Database, Table, and Column Information"](#).

2.3.13. Troubleshooting a MySQL Installation Under Windows

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. The purpose of this section is to help you diagnose and correct some of these errors.

Your first resource when troubleshooting server issues is the error log. The MySQL server uses the error log to record information relevant to the error that prevents the server from starting. The error log is located in the data directory specified in your `my.ini` file. The default data directory location is `C:\Program Files\MySQL\MySQL Server 5.0\data`. See [Section 5.12.1, “The Error Log”](#).

Another source of information regarding possible errors is the console messages displayed when the MySQL service is starting. Use the **NET START MySQL** command from the command line after installing **mysqld** as a service to see any error messages regarding the starting of the MySQL server as a service. See [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

The following examples show other common error messages you may encounter when installing MySQL and starting the server for the first time:

- If the MySQL server cannot find the `mysql` privileges database or other critical files, you may see these messages:

```
System error 1067 has occurred.  
Fatal error: Can't open privilege tables: Table 'mysql.host' doe
```

These messages often occur when the MySQL base or data directories are installed in different locations than the default locations (`C:\Program Files\MySQL\MySQL Server 5.0` and `C:\Program Files\MySQL\MySQL Server 5.0\data`, respectively).

This situation may occur when MySQL is upgraded and installed to a new location, but the configuration file is not updated to reflect the new location. In addition, there may be old and new configuration files that conflict. Be sure to delete or rename any old configuration files when upgrading MySQL.

If you have installed MySQL to a directory other than `C:\Program Files\MySQL\MySQL Server 5.0`, you need to ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. The `my.ini` file needs to be located in your Windows directory, typically `C:\WINDOWS` or `C:\WINNT`. You can determine its exact location from the value of the `WINDIR` environment variable by issuing the following command from the command prompt:

```
C:\> echo %WINDIR%
```

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in E:\mysql and the data directory is D:\MySQLdata, you can create the option file and set up a [mysqld] section to specify values for the basedir and datadir parameters:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

Note that Windows pathnames are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, you must double them:

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\MySQL\\MySQL Server 5.0
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

If you change the datadir value in your MySQL configuration file, you must move the contents of the existing MySQL data directory before restarting the MySQL server.

See [Section 2.3.7, “Creating an Option File”](#).

- If you reinstall or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Configuration Wizard, you may see this error:

```
Error: Cannot create Windows service for MySql. Error: 0
```

This occurs when the Configuration Wizard tries to install the service and finds an existing service with the same name.

One solution to this problem is to choose a service name other than mysql when using the configuration wizard. This allows the new service to be installed correctly, but leaves the outdated service in place. Although this is harmless, it is best to remove old services that are no longer in use.

To permanently remove the old `mysql` service, execute the following command as a user with administrative privileges, on the command-line:

```
C:\> sc delete mysql
[SC] DeleteService SUCCESS
```

If the `sc` utility is not available for your version of Windows, download the `delsrv` utility from <http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv.o.asp> and use the `delsrv mysql` syntax.

2.3.14. Upgrading MySQL on Windows

This section lists some of the steps you should take when upgrading MySQL on Windows.

1. Review [Section 2.11, “Upgrading MySQL”](#), for additional information on upgrading MySQL that is not specific to Windows.
2. You should always back up your current MySQL installation before performing an upgrade. See [Section 5.10.1, “Database Backups”](#).
3. Download the latest Windows distribution of MySQL from <http://dev.mysql.com/downloads/>.
4. Before upgrading MySQL, you must stop the server. If the server is installed as a service, stop the service with the following command from the command prompt:

```
C:\> NET STOP MySQL
```

If you are not running the MySQL server as a service, use the following command to stop it:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin" -u
```

Note: If the MySQL root user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

5. When upgrading to MySQL 5.0 from a version previous to 4.1.5, or when upgrading from a version of MySQL installed from a Zip archive to a

version of MySQL installed with the MySQL Installation Wizard, you must manually remove the previous installation and MySQL service (if the server is installed as a service).

To remove the MySQL service, use the following command:

```
C:\> C:\mysql\bin\mysqld --remove
```

If you do not remove the existing service, the MySQL Installation Wizard may fail to properly install the new MySQL service.

6. If you are using the MySQL Installation Wizard, start the wizard as described in [Section 2.3.3, “Using the MySQL Installation Wizard”](#).
7. If you are installing MySQL from a Zip archive, extract the archive. You may either overwrite your existing MySQL installation (usually located at C:\mysql), or install it into a different directory, such as C:\mysql5. Overwriting the existing installation is recommended.
8. If you were running MySQL as a Windows service and you had to remove the service earlier in this procedure, reinstall the service. (See [Section 2.3.11, “Starting MySQL as a Windows Service”](#).)
9. Restart the server. For example, use **NET START MySQL** if you run MySQL as a service, or invoke **mysqld** directly otherwise.
10. If you encounter errors, see [Section 2.3.13, “Troubleshooting a MySQL Installation Under Windows”](#).

2.3.15. MySQL on Windows Compared to MySQL on Unix

MySQL for Windows has proven itself to be very stable. The Windows version of MySQL has the same features as the corresponding Unix version, with the following exceptions:

- **Windows 95 and threads**

Windows 95 leaks about 200 bytes of main memory for each thread creation. Each connection in MySQL creates a new thread, so you shouldn't run **mysqld** for an extended time on Windows 95 if your server handles

many connections! Newer versions of Windows don't suffer from this bug.

- **Limited number of ports**

Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Note that ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

For more information about this problem, see

<http://support.microsoft.com/default.aspx?scid=kb;en-us;196271>.

- **Concurrent reads**

MySQL depends on the `pread()` and `pwrite()` system calls to be able to mix `INSERT` and `SELECT`. Currently, we use mutexes to emulate `pread()` and `pwrite()`. We intend to replace the file level interface with a virtual interface in the future so that we can use the `readfile()/writefile()` interface on NT, 2000, and XP to get more speed. The current implementation limits the number of open files that MySQL 5.0 can use to 2,048, which means that you cannot run as many concurrent threads on Windows NT, 2000, XP, and 2003 as on Unix.

- **Blocking read**

MySQL uses a blocking read for each connection. That has the following implications if named-pipe connections are enabled:

- A connection is not disconnected automatically after eight hours, as happens with the Unix version of MySQL.
- If a connection hangs, it is not possible to break it without killing MySQL.
- **mysqladmin kill** does not work on a sleeping connection.

- **mysqladmin shutdown** cannot abort as long as there are sleeping connections.

We plan to fix this problem in the future.

- **ALTER TABLE**

While you are executing an `ALTER TABLE` statement, the table is locked from being used by other threads. This has to do with the fact that on Windows, you can't delete a file that is in use by another thread. In the future, we may find some way to work around this problem.

- **DROP TABLE**

`DROP TABLE` on a table that is in use by a MERGE table does not work on Windows because the MERGE handler does the table mapping hidden from the upper layer of MySQL. Because Windows does not allow dropping files that are open, you first must flush all MERGE tables (with `FLUSH TABLES`) or drop the MERGE table before dropping the table.

- **DATA DIRECTORY and INDEX DIRECTORY**

The `DATA DIRECTORY` and `INDEX DIRECTORY` options for `CREATE TABLE` are ignored on Windows, because Windows doesn't support symbolic links. These options also are ignored on systems that have a non-functional `realpath()` call.

- **DROP DATABASE**

You cannot drop a database that is in use by some thread.

- **Killing MySQL from the Task Manager**

On Windows 95, you cannot kill MySQL from the Task Manager or with the shutdown utility. You must stop it with **mysqladmin shutdown**.

- **Case-insensitive names**

Filenames are not case sensitive on Windows, so MySQL database and table names are also not case sensitive on Windows. The only restriction is

that database and table names must be specified using the same case throughout a given statement. See [Section 9.2.2, “Identifier Case Sensitivity”](#).

- **The ‘\’ pathname separator character**

Pathname components in Windows are separated by the ‘\’ character, which is also the escape character in MySQL. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, use Unix-style filenames with ‘/’ characters:

```
mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;  
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

Alternatively, you must double the ‘\’ character:

```
mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;  
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

- **Problems with pipes**

Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z / CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

This is mainly a problem when you try to apply a binary log as follows:

```
C:\> mysqlbinlog binary_log_file | mysql --user=root
```

If you have a problem applying the log and suspect that it is because of a `^Z / CHAR(24)` character, you can use the following workaround:

```
C:\> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql  
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

The latter command also can be used to reliably read in any SQL file that may contain binary data.

- **Access denied for user error**

If MySQL cannot resolve your hostname properly, you may get the following error when you attempt to run a MySQL client program to connect to a server running on the same machine:

```
Access denied for user 'some_user'@'unknown'  
to database 'mysql'
```

To fix this problem, you should create a file named `\windows\hosts` containing the following information:

```
127.0.0.1      localhost
```

Here are some open issues for anyone who might want to help us improve MySQL on Windows:

- Add macros to use the faster thread-safe increment/decrement methods provided by Windows.

2.4. Installing MySQL on Linux

The recommended way to install MySQL on Linux is by using the RPM packages. The MySQL RPMs are currently built on a SuSE Linux 7.3 system, but should work on most versions of Linux that support `rpm` and use `glibc`. To obtain RPM packages, see [Section 2.1.3, “How to Get MySQL”](#).

MySQL AB does provide some platform-specific RPMs; the difference between a platform-specific RPM and a generic RPM is that a platform-specific RPM is built on the targeted platform and is linked dynamically whereas a generic RPM is linked statically with `LinuxThreads`.

Note: RPM distributions of MySQL often are provided by other vendors. Be aware that they may differ in features and capabilities from those built by MySQL AB, and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead.

If you have problems with an RPM file (for example, if you receive the error `Sorry, the host 'xxxx' could not be looked up`), see [Section 2.13.1.2, “Linux Binary Distribution Notes”](#).

In most cases, you need to install only the `mysql-server` and `mysql-client` packages to get a functional MySQL installation. The other packages are not required for a standard installation. If you want to run a MySQL-Max server that has additional capabilities, you should also install the `mysql-max` RPM. However, you should do so only *after* installing the `mysql-server` RPM. See [Section 5.3, “The `mysqld-max` Extended MySQL Server”](#).

If you get a dependency failure when trying to install MySQL packages (for example, `error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...`), you should also install the `mysql-shared-compat` package, which includes both the shared libraries for backward compatibility (`libmysqlclient.so.12` for MySQL 4.0 and `libmysqlclient.so.10` for MySQL 3.23).

Some Linux distributions still ship with MySQL 3.23 and they usually link applications dynamically to save disk space. If these shared libraries are in a separate package (for example, `mysql-shared`), it is sufficient to simply leave

this package installed and just upgrade the MySQL server and client packages (which are statically linked and do not depend on the shared libraries). For distributions that include the shared libraries in the same package as the MySQL server (for example, Red Hat Linux), you could either install our 3.23 MySQL - shared RPM, or use the MySQL - shared - compat package instead. (Do not install both.)

The following RPM packages are available:

- MySQL - server - VERSION.i386.rpm

The MySQL server. You need this unless you only want to connect to a MySQL server running on another machine.

Note: Server RPM files were called MySQL - VERSION.i386.rpm before MySQL 4.0.10. That is, they did not have - server in the name.

- MySQL - Max - VERSION.i386.rpm

The MySQL - Max server. This server has additional capabilities that the one provided in the MySQL - server RPM does not. You must install the MySQL - server RPM first, because the MySQL - Max RPM depends on it.

- MySQL - client - VERSION.i386.rpm

The standard MySQL client programs. You probably always want to install this package.

- MySQL - bench - VERSION.i386.rpm

Tests and benchmarks. Requires Perl and the DBI and DBD : : mysql modules.

- MySQL - devel - VERSION.i386.rpm

The libraries and include files that are needed if you want to compile other MySQL clients, such as the Perl modules.

- MySQL - shared - VERSION.i386.rpm

This package contains the shared libraries (libmysqlclient . so*) that

certain languages and applications need to dynamically load and use MySQL. It contains single-threaded and thread-safe libraries. If you install this package, do not install the `MySQL-shared-compat` package.

- `MySQL-shared-compat-VERSION.i386.rpm`

This package includes the shared libraries for MySQL 3.23, 4.0, 4.1, and 5.0. It contains single-threaded and thread-safe libraries. Install this package instead of `MySQL-shared` if you have applications installed that are dynamically linked against older versions of MySQL but you want to upgrade to the current version without breaking the library dependencies.

- `MySQL-embedded-VERSION.i386.rpm`

The embedded MySQL server library (available as of MySQL 4.0).

- `MySQL-VERSION.src.rpm`

This contains the source code for all of the previous packages. It can also be used to rebuild the RPMs on other architectures (for example, Alpha or SPARC).

To see all files in an RPM package (for example, a `MySQL-server` RPM), run a command like this:

```
shell> rpm -qp1 MySQL-server-VERSION.i386.rpm
```

To perform a standard minimal installation, install the server and client RPMs:

```
shell> rpm -i MySQL-server-VERSION.i386.rpm
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

To install only the client programs, install just the client RPM:

```
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

RPM provides a feature to verify the integrity and authenticity of packages before installing them. If you would like to learn more about this feature, see [Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”](#).

The server RPM places data under the `/var/lib/mysql` directory. The RPM also

creates a login account for a user named `mysql` (if one does not exist) to use for running the MySQL server, and creates the appropriate entries in `/etc/init.d/` to start the server automatically at boot time. (This means that if you have performed a previous installation and have made changes to its startup script, you may want to make a copy of the script so that you don't lose it when you install a newer RPM.) See [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#), for more information on how MySQL can be started automatically on system startup.

If you want to install the MySQL RPM on older Linux distributions that do not support initialization scripts in `/etc/init.d` (directly or via a symlink), you should create a symbolic link that points to the location where your initialization scripts actually are installed. For example, if that location is `/etc/rc.d/init.d`, use these commands before installing the RPM to create `/etc/init.d` as a symbolic link that points there:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

However, all current major Linux distributions should support the new directory layout that uses `/etc/init.d`, because it is required for LSB (Linux Standard Base) compliance.

If the RPM files that you install include `mysql-server`, the **mysqld** server should be up and running after installation. You should be able to start using MySQL.

If something goes wrong, you can find more information in the binary installation section. See [Section 2.8, “Installing MySQL on Other Unix-Like Systems”](#).

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.5. Installing MySQL on Mac OS X

You can install MySQL on Mac OS X 10.3.x (“Panther”) or newer using a Mac OS X binary package in PKG format instead of the binary tarball distribution. Please note that older versions of Mac OS X (for example, 10.1.x or 10.2.x) are **not** supported by this package.

The package is located inside a disk image (.dmg) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.

To obtain MySQL, see [Section 2.1.3, “How to Get MySQL”](#).

Note: Before proceeding with the installation, be sure to shut down all running MySQL server instances by either using the MySQL Manager Application (on Mac OS X Server) or via **mysqladmin shutdown** on the command line.

To actually install the MySQL PKG file, double-click on the package icon. This launches the Mac OS X Package Installer, which guides you through the installation of MySQL.

Due to a bug in the Mac OS X package installer, you may see this error message in the destination disk selection dialog:

```
You cannot install this software on this disk. (null)
```

If this error occurs, simply click the Go Back button once to return to the previous screen. Then click Continue to advance to the destination disk selection again, and you should be able to choose the destination disk correctly. We have reported this bug to Apple and it is investigating this problem.

The Mac OS X PKG of MySQL installs itself into `/usr/local/mysql-VERSION` and also installs a symbolic link, `/usr/local/mysql`, that points to the new location. If a directory named `/usr/local/mysql` exists, it is renamed to `/usr/local/mysql.bak` first. Additionally, the installer creates the grant tables in the `mysql` database by executing **mysql_install_db**.

The installation layout is similar to that of a **tar** file binary distribution; all MySQL binaries are located in the directory `/usr/local/mysql/bin`. The

MySQL socket file is created as `/tmp/mysql.sock` by default. See [Section 2.1.5, “Installation Layouts”](#).

MySQL installation requires a Mac OS X user account named `mysql`. A user account with this name should exist by default on Mac OS X 10.2 and up.

If you are running Mac OS X Server, a version of MySQL should already be installed. The following table shows the versions of MySQL that ship with Mac OS X Server versions.

Mac OS X Server Version	MySQL Version
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16
10.4.0	4.1.10a

This manual section covers the installation of the official MySQL Mac OS X PKG only. Make sure to read Apple's help information about installing MySQL: Run the “Help View” application, select “Mac OS X Server” help, do a search for “MySQL,” and read the item entitled “Installing MySQL.”

For pre-installed versions of MySQL on Mac OS X Server, note especially that you should start `mysqld` with `safe_mysqld` instead of `mysqld_safe` if MySQL is older than version 4.0.

If you previously used Marc Liyanage's MySQL packages for Mac OS X from <http://www.entropy.ch>, you can simply follow the update instructions for packages using the binary installation layout as given on his pages.

If you are upgrading from Marc's 3.23.x versions or from the Mac OS X Server version of MySQL to the official MySQL PKG, you also need to convert the existing MySQL privilege tables to the current format, because some new security privileges have been added. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

If you want MySQL to start automatically during system startup, you also need to install the MySQL Startup Item. It is part of the Mac OS X installation disk

images as a separate installation package. Simply double-click the MySQLStartupItem.pkg icon and follow the instructions to install it. The Startup Item need be installed only once. There is no need to install it each time you upgrade the MySQL package later.

The Startup Item for MySQL is installed into /Library/StartupItems/MYSQLCOM. (Before MySQL 4.1.2, the location was /Library/StartupItems/MySQL, but that collided with the MySQL Startup Item installed by Mac OS X Server.) Startup Item installation adds a variable MYSQLCOM=-YES- to the system configuration file /etc/hostconfig. If you want to disable the automatic startup of MySQL, simply change this variable to MYSQLCOM=-NO-.

On Mac OS X Server, the default MySQL installation uses the variable MYSQL in the /etc/hostconfig file. The MySQL AB Startup Item installer disables this variable by setting it to MYSQL=-NO-. This avoids boot time conflicts with the MYSQLCOM variable used by the MySQL AB Startup Item. However, it does not shut down a running MySQL server. You should do that yourself.

After the installation, you can start up MySQL by running the following commands in a terminal window. You must have administrator privileges to perform this task.

If you have installed the Startup Item, use this command:

```
shell> sudo /Library/StartupItems/MYSQLCOM/MYSQLCOM start  
(Enter your password, if necessary)  
(Press Control-D or enter "exit" to exit the shell)
```

If you don't use the Startup Item, enter the following command sequence:

```
shell> cd /usr/local/mysql  
shell> sudo ./bin/mysqld_safe  
(Enter your password, if necessary)  
(Press Control-Z)  
shell> bg  
(Press Control-D or enter "exit" to exit the shell)
```

You should be able to connect to the MySQL server, for example, by running /usr/local/mysql/bin/mysql.

Note: The accounts that are listed in the MySQL grant tables initially have no

passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as **mysql** and **mysqladmin** from the command line. The syntax for **bash** is:

```
alias mysql=/usr/local/mysql/bin/mysql
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

For **tcsch**, use:

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. For example, add the following line to your `$HOME/.bashrc` file if your shell is **bash**:

```
PATH=${PATH}:/usr/local/mysql/bin
```

Add the following line to your `$HOME/.tcshrc` file if your shell is **tcsch**:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If no `.bashrc` or `.tcshrc` file exists in your home directory, create it with a text editor.

If you are upgrading an existing installation, note that installing a new MySQL PKG does not remove the directory of an older installation. Unfortunately, the Mac OS X Installer does not yet offer the functionality required to properly upgrade previously installed packages.

To use your existing databases with the new installation, you'll need to copy the contents of the old data directory to the new data directory. Make sure that neither the old server nor the new one is running when you do this. After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-VERSION.pkg`.

2.6. Installing MySQL on Solaris

If you install MySQL using a binary tarball distribution on Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris **tar** cannot handle long filenames. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU **tar** (**gtar**) to unpack the distribution. You can find a precompiled copy for Solaris at <http://dev.mysql.com/downloads/os-solaris.html>.

You can install MySQL on Solaris using a binary package in PKG format instead of the binary tarball distribution. Before installing using the binary PKG format, you should create the `mysql` user and group, for example:

```
groupadd mysql
useradd -g mysql mysql
```

Some basic PKG-handling commands follow:

- To add a package:

```
pkgadd -d package_name.pkg
```

- To remove a package:

```
pkgrm package_name
```

- To get a full list of installed packages:

```
pkginfo
```

- To get detailed information for a package:

```
pkginfo -l package_name
```

- To list the files belonging to a package:

```
pkgchk -v package_name
```

- To get packaging information for an arbitrary file:

```
pkgchk -l -p file_name
```

For additional information about installing MySQL on Solaris, see [Section 2.13.3, “Solaris Notes”](#).

2.7. Installing MySQL on NetWare

Porting MySQL to NetWare was an effort spearheaded by Novell. Novell customers should be pleased to note that NetWare 6.5 ships with bundled MySQL binaries, complete with an automatic commercial use license for all servers running that version of NetWare.

MySQL for NetWare is compiled using a combination of Metrowerks CodeWarrior for NetWare and special cross-compilation versions of the GNU autotools.

The latest binary packages for NetWare can be obtained at <http://dev.mysql.com/downloads/>. See [Section 2.1.3, “How to Get MySQL”](#).

To host MySQL, the NetWare server must meet these requirements:

- The latest Support Pack of [NetWare 6.5](#) must be installed.
- The system must meet Novell's minimum requirements to run the respective version of NetWare.
- MySQL data and the program binaries must be installed on an NSS volume; traditional volumes are not supported.

To install MySQL for NetWare, use the following procedure:

1. If you are upgrading from a prior installation, stop the MySQL server. This is done from the server console, using the following command:

```
SERVER: mysqladmin -u root shutdown
```

Note: If the MySQL root user account has a password, you need to invoke **mysqladmin** with the **-p** option and supply the password when prompted.

2. Log on to the target server from a client machine with access to the location where you are installing MySQL.
3. Extract the binary package Zip file onto the server. Be sure to allow the paths in the Zip file to be used. It is safe to simply extract the file to `SYS:\`.

If you are upgrading from a prior installation, you may need to copy the data directory (for example, `SYS:MYSQL\DATA`), as well as `my.cnf`, if you have customized it. You can then delete the old copy of MySQL.

4. You might want to rename the directory to something more consistent and easy to use. The examples in this manual use `SYS:MYSQL` to refer to the installation directory.

Note that MySQL installation on NetWare does not detect if a version of MySQL is already installed outside the NetWare release. Therefore, if you have installed the latest MySQL version from the Web (for example, MySQL 4.1 or later) in `SYS:\MYSQL`, you must rename the folder before upgrading the NetWare server; otherwise, files in `SYS:\MYSQL` are overwritten by the MySQL version present in NetWare Support Pack.

5. At the server console, add a search path for the directory containing the MySQL NLMs. For example:

```
SERVER:  SEARCH ADD SYS:MYSQL\BIN
```

6. Initialize the data directory and the grant tables, if necessary, by executing **mysql_install_db** at the server console.
7. Start the MySQL server using **mysqld_safe** at the server console.
8. To finish the installation, you should also add the following commands to `autoexec.ncf`. For example, if your MySQL installation is in `SYS:MYSQL` and you want MySQL to start automatically, you could add these lines:

```
#Starts the MySQL 5.0.x database server  
SEARCH ADD SYS:MYSQL\BIN  
MYSQLD_SAFE
```

If you are running MySQL on NetWare 6.0, we strongly suggest that you use the `--skip-external-locking` option on the command line:

```
#Starts the MySQL 5.0.x database server  
SEARCH ADD SYS:MYSQL\BIN  
MYSQLD_SAFE --skip-external-locking
```

It is also necessary to use `CHECK TABLE` and `REPAIR TABLE` instead of **myisamchk**, because **myisamchk** makes use of external locking. External

locking is known to have problems on NetWare 6.0; the problem has been eliminated in NetWare 6.5. Note that the use of MySQL on Netware 6.0 is not officially supported.

mysqld_safe on NetWare provides a screen presence. When you unload (shut down) the **mysqld_safe** NLM, the screen does not go away by default. Instead, it prompts for user input:

```
*<NLM has terminated; Press any key to close the screen>*
```

If you want NetWare to close the screen automatically instead, use the `--autoclose` option to **mysqld_safe**. For example:

```
#Starts the MySQL 5.0.x database server
SEARCH ADD SYS:MYSQL\BIN
MYSQLD_SAFE --autoclose
```

The behavior of **mysqld_safe** on NetWare is described further in [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#).

9. When installing MySQL, either for the first time or upgrading from a previous version, download and install the latest and appropriate Perl module and PHP extensions for NetWare:
 - Perl:
<http://forge.novell.com/modules/xfcontent/downloads.php/perl/Module>
 - PHP:
<http://forge.novell.com/modules/xfcontent/downloads.php/php/Module>

If there was an existing installation of MySQL on the NetWare server, be sure to check for existing MySQL startup commands in `autoexec.ncf`, and edit or delete them as necessary.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.8. Installing MySQL on Other Unix-Like Systems

This section covers the installation of MySQL binary distributions that are provided for various platforms in the form of compressed **tar** files (files with a `.tar.gz` extension). See [Section 2.1.2.5, “MySQL Binaries Compiled by MySQL AB”](#), for a detailed list.

To obtain MySQL, see [Section 2.1.3, “How to Get MySQL”](#).

MySQL **tar** file binary distributions have names of the form `mysql-VERSION-OS.tar.gz`, where `VERSION` is a number (for example, `5.0.25`), and `OS` indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686`).

In addition to these generic packages, we also offer binaries in platform-specific package formats for selected platforms. See [Section 2.2, “Standard MySQL Installation Using a Binary Distribution”](#), for more information on how to install these.

You need the following tools to install a MySQL **tar** file binary distribution:

- GNU `gunzip` to uncompress the distribution.
- A reasonable **tar** to unpack the distribution. GNU **tar** is known to work. Some operating systems come with a pre-installed version of **tar** that is known to have problems. For example, Mac OS X **tar** and Sun **tar** are known to have problems with long filenames. On Mac OS X, you can use the pre-installed **gnutar** program. On other systems with a deficient **tar**, you should install GNU **tar** first.

If you run into problems and need to file a bug report, please use the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

The basic commands that you must execute to install and use a MySQL binary distribution are:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /usr/local
```

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

Note: This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to [Section 2.10, “Post-Installation Setup and Testing”](#).

A more detailed version of the preceding description for installing a binary distribution follows:

1. Add a login user and group for **mysqld** to run as:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

These commands add the `mysql` group and the `mysql` user. The syntax for **useradd** and **groupadd** may differ slightly on different versions of Unix, or they may have different names such as **adduser** and **addgroup**.

You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Pick the directory under which you want to unpack the distribution and change location into it. In the following example, we unpack the distribution under `/usr/local`. (The instructions, therefore, assume that you have permission to create files and directories in `/usr/local`. If that directory is protected, you must perform the installation as root.)

```
shell> cd /usr/local
```

3. Obtain a distribution file using the instructions in [Section 2.1.3, “How to Get MySQL”](#). For a given release, binary distributions for all platforms are built from the same MySQL source distribution.
4. Unpack the distribution, which creates the installation directory. Then create a symbolic link to that directory:

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

The **tar** command creates a directory named `mysql-VERSION-OS`. The `ln` command makes a symbolic link to that directory. This lets you refer more easily to the installation directory as `/usr/local/mysql`.

With GNU **tar**, no separate invocation of `gunzip` is necessary. You can replace the first line with the following alternative command to uncompress and extract the distribution:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

5. Change location into the installation directory:

```
shell> cd mysql
```

You will find several files and subdirectories in the `mysql` directory. The most important for installation purposes are the `bin` and `scripts` subdirectories:

- The `bin` directory contains client programs and the server. You should add the full pathname of this directory to your `PATH` environment variable so that your shell finds the MySQL programs properly. See [Appendix F, *Environment Variables*](#).
- The `scripts` directory contains the **`mysql_install_db`** script used to initialize the `mysql` database containing the grant tables that store the server access permissions.

6. If you have not installed MySQL before, you must create the MySQL grant tables:

```
shell> scripts/mysql_install_db --user=mysql
```

If you run the command as `root`, you must use the `--user` option as shown. The value of the option should be the name of the login account that you created in the first step to use for running the server. If you run the command while logged in as that user, you can omit the `--user` option.

After creating or updating the grant tables, you need to restart the server

manually.

7. Change the ownership of program binaries to root and ownership of the data directory to the user that you run **mysqld** as. Assuming that you are located in the installation directory (`/usr/local/mysql`), the commands look like this:

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

The first command changes the owner attribute of the files to the root user. The second changes the owner attribute of the data directory to the mysql user. The third changes the group attribute to the mysql group.

8. If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself and in [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).
9. You can set up new accounts using the `bin/mysql_setpermission` script if you install the DBI and DBD::mysql Perl modules. For instructions, see [Section 2.14, “Perl Installation Notes”](#).
10. If you would like to use **mysqlaccess** and have the MySQL distribution in some non-standard location, you must change the location where **mysqlaccess** expects to find the **mysql** client. Edit the `bin/mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL      = '/usr/local/bin/mysql';    # path to mysql executat
```

Change the path to reflect the location where **mysql** actually is stored on your system. If you do not do this, a Broken pipe error will occur when you run **mysqlaccess**.

After everything has been unpacked and installed, you should test your distribution. To start the MySQL server, use the following command:

```
shell> bin/mysqld_safe --user=mysql &
```

If that command fails immediately and prints `mysqld ended`, you can find some information in the `host_name.err` file in the data directory.

More information about **`mysqld_safe`** is given in [Section 5.4.1, “`mysqld_safe` — MySQL Server Startup Script](#)”.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing](#)”.

2.9. MySQL Installation Using a Source Distribution

Before you proceed with an installation from source, first check whether our binary is available for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options.

To obtain a source distribution for MySQL, [Section 2.1.3, “How to Get MySQL”](#).

MySQL source distributions are provided as compressed **tar** archives and have names of the form `mysql-VERSION.tar.gz`, where *VERSION* is a number like `5.0.25`.

You need the following tools to build and install MySQL from source:

- GNU **gunzip** to uncompress the distribution.
- A reasonable **tar** to unpack the distribution. GNU **tar** is known to work. Some operating systems come with a pre-installed version of **tar** that is known to have problems. For example, the **tar** provided with early versions of Mac OS X **tar**, SunOS 4.x and Solaris 8 and earlier are known to have problems with long filenames. On Mac OS X, you can use the pre-installed **gnutar** program. On other systems with a deficient **tar**, you should install GNU **tar** first.
- A working ANSI C++ compiler. **gcc** 2.95.2 or later, **egcs** 1.0.2 or later or **egcs 2.91.66**, SGI C++, and SunPro C++ are some of the compilers that are known to work. `libg++` is not needed when using **gcc**. **gcc** 2.7.x has a bug that makes it impossible to compile some perfectly legal C++ files, such as `sql/sql_base.cc`. If you have only **gcc** 2.7.x, you must upgrade your **gcc** to be able to compile MySQL. **gcc** 2.8.1 is also known to have problems on some platforms, so it should be avoided if a new compiler exists for the platform.

gcc 2.95.2 or later is recommended when compiling MySQL 3.23.x.

- A good **make** program. GNU **make** is always recommended and is

sometimes required. If you have problems, we recommend GNU **make** 3.75 or newer.

If you are using a version of **gcc** recent enough to understand the `-fno-exceptions` option, it is *very important* that you use this option. Otherwise, you may compile a binary that crashes randomly. We also recommend that you use `-felide-constructors` and `-fno-rtti` along with `-fno-exceptions`. When in doubt, do the following:

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \  
-fno-exceptions -fno-rtti" ./configure \  
--prefix=/usr/local/mysql --enable-assembly \  
--with-mysqld-ldflags=-all-static
```

On most systems, this gives you a fast and stable binary.

If you run into problems and need to file a bug report, please use the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

2.9.1. Source Installation Overview

The basic commands that you must execute to install a MySQL source distribution are:

```
shell> groupadd mysql  
shell> useradd -g mysql mysql  
shell> gunzip < mysql-VERSION.tar.gz | tar -xvf -  
shell> cd mysql-VERSION  
shell> ./configure --prefix=/usr/local/mysql  
shell> make  
shell> make install  
shell> cp support-files/my-medium.cnf /etc/my.cnf  
shell> cd /usr/local/mysql  
shell> bin/mysql_install_db --user=mysql  
shell> chown -R root .  
shell> chown -R mysql var  
shell> chgrp -R mysql .  
shell> bin/mysqld_safe --user=mysql &
```

If you start from a source RPM, do the following:

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

This makes a binary RPM that you can install. For older versions of RPM, you

may have to replace the command **rpmbuild** with **rpm** instead.

Note: This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to [Section 2.10, “Post-Installation Setup and Testing”](#), for post-installation setup and testing.

A more detailed version of the preceding description for installing MySQL from a source distribution follows:

1. Add a login user and group for **mysqld** to run as:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

These commands add the `mysql` group and the `mysql` user. The syntax for **useradd** and **groupadd** may differ slightly on different versions of Unix, or they may have different names such as **adduser** and **addgroup**.

You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Pick the directory under which you want to unpack the distribution and change location into it.
3. Obtain a distribution file using the instructions in [Section 2.1.3, “How to Get MySQL”](#).
4. Unpack the distribution into the current directory:

```
shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
```

This command creates a directory named `mysql-VERSION`.

With GNU **tar**, no separate invocation of `gunzip` is necessary. You can use the following alternative command to uncompress and extract the distribution:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

5. Change location into the top-level directory of the unpacked distribution:

```
shell> cd mysql-VERSION
```

Note that currently you must configure and build MySQL from this top-level directory. You cannot build it in a different directory.

6. Configure the release and compile everything:

```
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

When you run **configure**, you might want to specify other options. Run **./configure --help** for a list of options. [Section 2.9.2, “Typical configure Options”](#), discusses some of the more useful options.

If **configure** fails and you are going to send mail to a MySQL mailing list to ask for assistance, please include any lines from `config.log` that you think can help solve the problem. Also include the last couple of lines of output from **configure**. To file a bug report, please use the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

If the compile fails, see [Section 2.9.4, “Dealing with Problems Compiling MySQL”](#), for help.

7. Install the distribution:

```
shell> make install
```

If you want to set up an option file, use one of those present in the `support-files` directory as a template. For example:

```
shell> cp support-files/my-medium.cnf /etc/my.cnf
```

You might need to run these commands as root.

If you want to configure support for InnoDB tables, you should edit the `/etc/my.cnf` file, remove the `#` character before the option lines that start with `innodb_...`, and modify the option values to be what you want. See [Section 4.3.2, “Using Option Files”](#), and [Section 14.2.3, “InnoDB Configuration”](#).

8. Change location into the installation directory:

```
shell> cd /usr/local/mysql
```

9. If you haven't installed MySQL before, you must create the MySQL grant tables:

```
shell> bin/mysql_install_db --user=mysql
```

If you run the command as `root`, you should use the `--user` option as shown. The value of the option should be the name of the login account that you created in the first step to use for running the server. If you run the command while logged in as that user, you can omit the `--user` option.

After using `mysql_install_db` to create the grant tables for MySQL, you must restart the server manually. The `mysqld_safe` command to do this is shown in a later step.

10. Change the ownership of program binaries to `root` and ownership of the data directory to the user that you run `mysqld` as. Assuming that you are located in the installation directory (`/usr/local/mysql`), the commands look like this:

```
shell> chown -R root .
shell> chown -R mysql var
shell> chgrp -R mysql .
```

The first command changes the owner attribute of the files to the `root` user. The second changes the owner attribute of the data directory to the `mysql` user. The third changes the group attribute to the `mysql` group.

11. If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself; see also [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).
12. You can set up new accounts using the `bin/mysql_setpermission` script if you install the `DBI` and `DBD: :mysql` Perl modules. For instructions, see [Section 2.14, “Perl Installation Notes”](#).

After everything has been installed, you should test your distribution. To start the MySQL server, use the following command:

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

If that command fails immediately and prints `mysqld ended`, you can find some information in the `host_name.err` file in the data directory.

More information about `mysqld_safe` is given in [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#).

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in [Section 2.10, “Post-Installation Setup and Testing”](#).

2.9.2. Typical configure Options

The `configure` script gives you a great deal of control over how you configure a MySQL source distribution. Typically you do this using options on the `configure` command line. You can also affect `configure` using certain environment variables. See [Appendix F, Environment Variables](#). For a list of options supported by `configure`, run this command:

```
shell> ./configure --help
```

Some of the more commonly used `configure` options are described here:

- To compile just the MySQL client libraries and client programs and not the server, use the `--without-server` option:

```
shell> ./configure --without-server
```

If you have no C++ compiler, some client programs such as `mysql` cannot be compiled because they require C++. In this case, you can remove the code in `configure` that tests for the C++ compiler and then run `./configure` with the `--without-server` option. The compile step should still try to build all clients, but you can ignore any warnings about files such as `mysql.cc`. (If `make` stops, try `make -k` to tell it to continue with the rest of the build even if errors occur.)

- If you want to build the embedded MySQL library (`libmysqld.a`), use the `--with-embedded-server` option.
- If you don't want your log files and database directories located under `/usr/local/var`, use a `configure` command something like one of these:

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
    --localstatedir=/usr/local/mysql/data
```

The first command changes the installation prefix so that everything is installed under `/usr/local/mysql` rather than the default of `/usr/local`. The second command preserves the default installation prefix, but overrides the default location for database directories (normally `/usr/local/var`) and changes it to `/usr/local/mysql/data`.

You can also specify the installation directory and data directory locations at server startup time by using the `--basedir` and `--datadir` options. These can be given on the command line or in an MySQL option file, although it is more common to use an option file. See [Section 4.3.2, “Using Option Files”](#).

- If you are using Unix and you want the MySQL socket file location to be somewhere other than the default location (normally in the directory `/tmp` or `/var/run`), use a **configure** command like this:

```
shell> ./configure \
    --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

The socket filename must be an absolute pathname. You can also change the location of `mysql.sock` at server startup by using a MySQL option file. See [Section A.4.5, “How to Protect or Change the MySQL Unix Socket File”](#).

- If you want to compile statically linked programs (for example, to make a binary distribution, to get better performance, or to work around problems with some Red Hat Linux distributions), run **configure** like this:

```
shell> ./configure --with-client-ldflags=-all-static \
    --with-mysqld-ldflags=-all-static
```

- If you are using **gcc** and don't have `libg++` or `libstdc++` installed, you can tell **configure** to use **gcc** as your C++ compiler:

```
shell> CC=gcc CXX=gcc ./configure
```

When you use **gcc** as your C++ compiler, it does not attempt to link in `libg++` or `libstdc++`. This may be a good thing to do even if you have

those libraries installed. Some versions of them have caused strange problems for MySQL users in the past.

The following list indicates some compilers and environment variable settings that are commonly used with each one.

- **gcc 2.7.2:**

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
```

- **egcs 1.0.3a:**

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors \  
-fno-exceptions -fno-rtti"
```

- **gcc 2.95.2:**

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \  
-felide-constructors -fno-exceptions -fno-rtti"
```

- **pgcc 2.90.29 or newer:**

```
CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc \  
CXXFLAGS="-O3 -mpentiumpro -mstack-align-double \  
-felide-constructors -fno-exceptions -fno-rtti"
```

In most cases, you can get a reasonably optimized MySQL binary by using the options from the preceding list and adding the following options to the **configure** line:

```
--prefix=/usr/local/mysql --enable-asm --enable-asm-asm \\  
--with-mysqld-ldflags=-all-static
```

The full **configure** line would, in other words, be something like the following for all recent **gcc** versions:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \  
-felide-constructors -fno-exceptions -fno-rtti" ./configure \  
--prefix=/usr/local/mysql --enable-asm --enable-asm-asm \  
--with-mysqld-ldflags=-all-static
```

The binaries we provide on the MySQL Web site at <http://dev.mysql.com/downloads/> are all compiled with full optimization and should be perfect for most users. See [Section 2.1.2.5, “MySQL Binaries](#)

[Compiled by MySQL AB](#)". There are some configuration settings you can tweak to build an even faster binary, but these are only for advanced users. See [Section 7.5.4, "How Compiling and Linking Affects the Speed of MySQL"](#).

If the build fails and produces errors about your compiler or linker not being able to create the shared library `libmysqlclient.so.N` (where *N* is a version number), you can work around this problem by giving the `--disable-shared` option to **configure**. In this case, **configure** does not build a shared `libmysqlclient.so.N` library.

- By default, MySQL uses the `latin1` (cp1252 West European) character set. To change the default set, use the `--with-charset` option:

```
shell> ./configure --with-charset=CHARSET
```

CHARSET may be one of `big5`, `cp1251`, `cp1257`, `czech`, `danish`, `dec8`, `dos`, `euc_kr`, `gb2312`, `gbk`, `german1`, `hebrew`, `hp8`, `hungarian`, `koi8_ru`, `koi8_ukr`, `latin1`, `latin2`, `sjis`, `swe7`, `tis620`, `ujis`, `usa7`, or `win1251ukr`. See [Section 5.11.1, "The Character Set Used for Data and Sorting"](#). (Additional character sets might be available. Check the output from `./configure --help` for the current list.)

The default collation may also be specified. MySQL uses the `latin1_swedish_ci` collation by default. To change this, use the `--with-collation` option:

```
shell> ./configure --with-collation=COLLATION
```

To change both the character set and the collation, use both the `--with-charset` and `--with-collation` options. The collation must be a legal collation for the character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.)

Warning: If you change character sets after having created any tables, you must run `myisamchk -r -q --set-collation=collation_name` on every *MyISAM* table. Your indexes may be sorted incorrectly otherwise. This can happen if you install MySQL, create some tables, and then reconfigure MySQL to use a different character set and reinstall it.

With the **configure** option `--with-extra-charsets=LIST`, you can define which additional character sets should be compiled into the server. *LIST* is one of the following:

- A list of character set names separated by spaces
- `complex` to include all character sets that can't be dynamically loaded
- `all` to include all character sets into the binaries

Clients that want to convert characters between the server and the client should use the `SET NAMES` statement. See [Section 13.5.3, “SET Syntax”](#), and [Section 10.4, “Connection Character Sets and Collations”](#).

- To configure MySQL with debugging code, use the `--with-debug` option:

```
shell> ./configure --with-debug
```

This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. See [Section E.1, “Debugging a MySQL Server”](#).

- If your client programs are using threads, you must compile a thread-safe version of the MySQL client library with the `--enable-thread-safe-client` configure option. This creates a `libmysqlclient_r` library with which you should link your threaded applications. See [Section 22.2.15, “How to Make a Threaded Client”](#).
- It is possible to build MySQL 5.0 with large table support using the `--with-big-tables` option, beginning with MySQL 5.0.4.

This option causes the variables that store table row counts to be declared as `unsigned long long` rather than `unsigned long`. This enables tables to hold up to approximately $1.844\text{E}+19$ ($(2^{32})^2$) rows rather than 2^{32} ($\sim 4.295\text{E}+09$) rows. Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually in order to enable this feature.

- See [Section 2.13, “Operating System-Specific Notes”](#), for options that pertain to particular operating systems.

- See [Section 5.9.7.2, “Using SSL Connections”](#), for options that pertain to configuring MySQL to support secure (encrypted) connections.

2.9.3. Installing from the Development Source Tree

Caution: You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL up and running on your system, you should use a standard release distribution (either a binary or source distribution).

To obtain our most recent development source tree, first download and install the BitKeeper free client if you do not have it. The client can be obtained from <http://www.bitmover.com/bk-client.shar>.

To install the BitKeeper client on Unix, use these commands:

```
shell> sh bk-client.shar
shell> cd bk_client-1.1
shell> make all
shell> PATH=$PWD:$PATH
```

To install the BitKeeper client on Windows, use these instructions:

1. Download and install Cygwin from <http://cygwin.com>.
2. Make sure **gcc** and **make** have been installed under Cygwin. You can test this by issuing **which gcc** and **which make** commands. If either one is not installed, run Cygwin's package manager, select **gcc**, **make**, or both, and install them.
3. Under Cygwin, execute these commands:

```
shell> sh bk-client.shar
shell> cd bk_client-1.1
```

Then edit the Makefile and change the line that reads `$(CC) $(CFLAGS) -o sfio -lz sfio.c` to this:

```
$(CC) $(CFLAGS) -o sfio sfio.c -lz
```

Now run the **make** command and set the path:

```
shell> make all
shell> PATH=$PWD:$PATH
```

The BitKeeper free client is shipped with its source code. The only documentation available for the free client is the source code itself.

After you have installed the BitKeeper client, you can access the MySQL development source tree:

1. Change location to the directory you want to work from, and then use the following command to make a local copy of the MySQL 5.0 branch:

```
shell> sfioball -r+ bk://mysql.bkbits.net/mysql-5.0 mysql-5.0
```

In the preceding example, the source tree is set up in the `mysql-5.0/` subdirectory of your current directory.

The initial download of the source tree may take a while, depending on the speed of your connection. Please be patient.

2. You need GNU **make**, **autoconf** 2.58 (or newer), **automake** 1.8, **libtool** 1.5, and **m4** to run the next set of commands. Even though many operating systems come with their own implementation of **make**, chances are high that the compilation fails with strange error messages. Therefore, it is highly recommended that you use GNU **make** (sometimes named **gmake**) instead.

Fortunately, a large number of operating systems ship with the GNU toolchain preinstalled or supply installable packages of these. In any case, they can also be downloaded from the following locations:

- <http://www.gnu.org/software/autoconf/>
- <http://www.gnu.org/software/automake/>
- <http://www.gnu.org/software/libtool/>
- <http://www.gnu.org/software/m4/>
- <http://www.gnu.org/software/make/>

To configure MySQL 5.0, you also need GNU **bison** 1.75 or later. Older versions of **bison** may report this error:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceed
```

Note: The maximum table size is not actually exceeded; the error is caused by bugs in older versions of **bison**.

The following example shows the typical commands required to configure a source tree. The first `cd` command changes location into the top-level directory of the tree; replace `mysql-5.0` with the appropriate directory name.

```
shell> cd mysql-5.0
shell> (cd bdb/dist; sh s_all)
shell> (cd innobase; autoreconf --force --install)
shell> autoreconf --force --install
shell> ./configure # Add your favorite options here
shell> make
```

Or you can use **BUILD/autorun.sh** as a shortcut for the following sequence of commands:

```
shell> aclocal; autoheader
shell> libtoolize --automake --force
shell> automake --force --add-missing; autoconf
shell> (cd innobase; aclocal; autoheader; autoconf; automake)
shell> (cd bdb/dist; sh s_all)
```

The command lines that change directory into the `innobase` and `bdb/dist` directories are used to configure the InnoDB and Berkeley DB (BDB) storage engines. You can omit these command lines if you do not require InnoDB or BDB support.

If you get some strange errors during this stage, verify that you really have **libtool** installed.

A collection of our standard configuration scripts is located in the `BUILD/` subdirectory. You may find it more convenient to use the `BUILD/compile-pentium-debug` script than the preceding set of shell commands. To compile on a different architecture, modify the script by removing flags that are Pentium-specific.

3. When the build is done, run **make install**. Be careful with this on a production machine; the command may overwrite your live release installation. If you have another installation of MySQL, we recommend that you run **./configure** with different values for the `--prefix`, `--with-tcp-port`, and `--unix-socket-path` options than those used for your production server.
4. Play hard with your new installation and try to make the new features crash. Start by running **make test**. See [Section 24.1.2, “MySQL Test Suite”](#).
5. If you have gotten to the **make** stage, but the distribution does not compile, please enter the problem into our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#). If you have installed the latest versions of the required GNU tools, and they crash trying to process our configuration files, please report that also. However, if you execute `aclocal` and get a command not found error or a similar problem, do not report it. Instead, make sure that all the necessary tools are installed and that your `PATH` variable is set correctly so that your shell can find them.
6. After initially copying the repository with **sfio**ball to obtain the source tree, you should use **update** periodically to update your local copy. To do this any time after you have set up the repository, use this command:

```
shell> update bk://mysql.bkbits.net/mysql-5.0
```
7. You can examine the change history for the tree with all the diffs by viewing the `BK/ChangeLog` file in the source tree and looking at the `ChangeSet` descriptions listed there. To examine a particular changeset, you would have to use the **sfio**ball command to extract two particular revisions of the source tree, and then use an external **diff** command to compare them. If you see some funny diffs or code that you have a question about, do not hesitate to send email to the MySQL `internals` mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#). Also, if you think you have a better idea on how to do something, send an email message to the list with a patch.

You can also browse changesets, comments, and source code online. To browse this information for MySQL 5.0, go to <http://mysql.bkbits.net:8080/mysql-5.0>.

2.9.4. Dealing with Problems Compiling MySQL

All MySQL programs compile cleanly for us with no warnings on Solaris or Linux using **gcc**. On other systems, warnings may occur due to differences in system include files. See [Section 2.9.5, “MIT-pthreads Notes”](#), for warnings that may occur when using MIT-pthreads. For other problems, check the following list.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If **configure** is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `config.cache`. When **configure** starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run **configure**, you must run **make** again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before re-running **configure**:

```
shell> rm config.cache
shell> make clean
```

Alternatively, you can run **make distclean**.

The following list describes some of the problems when compiling MySQL that have been found to occur most often:

- If you get errors such as the ones shown here when compiling `sql_yacc.cc`, you probably have run out of memory or swap space:

```
Internal compiler error: program cc1plus got fatal signal 11
Out of virtual memory
Virtual memory exhausted
```

The problem is that **gcc** requires a huge amount of memory to compile `sql_yacc.cc` with inline functions. Try running **configure** with the `--with-`

low-memory option:

```
shell> ./configure --with-low-memory
```

This option causes `-fno-inline` to be added to the compile line if you are using `gcc` and `-O0` if you are using something else. You should try the `--with-low-memory` option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations, and the `--with-low-memory` option usually fixes it.

- By default, **configure** picks `c++` as the compiler name and GNU `c++` links with `-lg++`. If you are using `gcc`, that behavior can cause problems during configuration such as this:

```
configure: error: installation or configuration problem:  
C++ compiler cannot create executables.
```

You might also observe problems during compilation related to `g++`, `libg++`, or `libstdc++`.

One cause of these problems is that you may not have `g++`, or you may have `g++` but not `libg++`, or `libstdc++`. Take a look at the `config.log` file. It should contain the exact reason why your C++ compiler didn't work. To work around these problems, you can use `gcc` as your C++ compiler. Try setting the environment variable `CXX` to `"gcc -O3"`. For example:

```
shell> CXX="gcc -O3" ./configure
```

This works because `gcc` compiles C++ source files as well as `g++` does, but does not link in `libg++` or `libstdc++` by default.

Another way to fix these problems is to install `g++`, `libg++`, and `libstdc++`. However, we recommend that you not use `libg++` or `libstdc++` with MySQL because this only increases the binary size of **mysqld** without providing any benefits. Some versions of these libraries have also caused strange problems for MySQL users in the past.

- If your compile fails with errors such as any of the following, you must upgrade your version of **make** to GNU **make**:

```
making all in mit-pthreads
make: Fatal error in reader: Makefile, line 18:
Badly formed macro assignment
```

Or:

```
make: file `Makefile' line 18: Must be a separator (:
```

Or:

```
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome **make** programs.

GNU **make** 3.75 is known to work.

- If you want to define flags to be used by your C or C++ compilers, do so by adding the flags to the CFLAGS and CXXFLAGS environment variables. You can also specify the compiler names this way using CC and CXX. For example:

```
shell> CC=gcc
shell> CFLAGS=-O3
shell> CXX=gcc
shell> CXXFLAGS=-O3
shell> export CC CFLAGS CXX CXXFLAGS
```

See [Section 2.1.2.5, “MySQL Binaries Compiled by MySQL AB”](#), for a list of flag definitions that have been found to be useful on various systems.

- If you get errors such as those shown here when compiling **mysqld**, **configure** did not correctly detect the type of the last argument to `accept()`, `getsockname()`, or `getpeername()`:

```
cxx: Error: mysqld.cc, line 645: In this statement, the referenc
    type of the pointer value 'length' is 'unsigned long',
    which is not compatible with 'int'.
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
```

To fix this, edit the `config.h` file (which is generated by **configure**). Look for these lines:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Change `xxx` to `size_t` or `int`, depending on your operating system. (You must do this each time you run **configure** because **configure** regenerates `config.h`.)

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pre-generated copy. However, if you do need to re-create it, you might encounter this error:

```
"sql_yacc.yy", line xxx fatal: default action causes potential..
```

This is a sign that your version of **yacc** is deficient. You probably need to install **bison** (the GNU version of **yacc**) and use that instead.

- On Debian Linux 3.0, you need to install `gawk` instead of the default `mawk` if you want to compile MySQL with Berkeley DB support.
- If you need to debug **mysqld** or a MySQL client, run **configure** with the `--with-debug` option, and then recompile and link your clients with the new client library. See [Section E.2, “Debugging a MySQL Client”](#).
- If you get a compilation error on Linux (for example, SuSE Linux 8.1 or Red Hat Linux 7.3) similar to the following one, you probably do not have **g++** installed:

```
libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from
incompatible pointer type
libmysql.c:1329: too few arguments to function `gethostbyname_r'
libmysql.c:1329: warning: assignment makes pointer from integer
without a cast
make[2]: *** [libmysql.lo] Error 1
```

By default, the **configure** script attempts to determine the correct number of arguments by using **g++** (the GNU C++ compiler). This test yields incorrect results if **g++** is not installed. There are two ways to work around this problem:

- Make sure that the GNU C++ **g++** is installed. On some Linux distributions, the required package is called `gpp`; on others, it is named **gcc-c++**.

- Use **gcc** as your C++ compiler by setting the CXX environment variable to **gcc**:

```
export CXX="gcc"
```

You must run **configure** again after making either of those changes.

2.9.5. MIT-pthreads Notes

This section describes some of the issues involved in using MIT-pthreads.

On Linux, you should *not* use MIT-pthreads. Use the installed LinuxThreads implementation instead. See [Section 2.13.1, “Linux Notes”](#).

If your system does not provide native thread support, you should build MySQL using the MIT-pthreads package. This includes older FreeBSD systems, SunOS 4.x, Solaris 2.4 and earlier, and some others. See [Section 2.1.1, “Operating Systems Supported by MySQL”](#).

MIT-pthreads is not part of the MySQL 5.0 source distribution. If you require this package, you need to download it separately from http://www.mysql.com/Downloads/Contrib/pthreads-1_60_beta6-mysql.tar.gz

After downloading, extract this source archive into the top level of the MySQL source directory. It creates a new subdirectory named `mit-pthreads`.

- On most systems, you can force MIT-pthreads to be used by running **configure** with the `--with-mit-threads` option:

```
shell> ./configure --with-mit-threads
```

Building in a non-source directory is not supported when using MIT-pthreads because we want to minimize our changes to this code.

- The checks that determine whether to use MIT-pthreads occur only during the part of the configuration process that deals with the server code. If you have configured the distribution using `--without-server` to build only the client code, clients do not know whether MIT-pthreads is being used and use Unix socket file connections by default. Because Unix socket files do not work under MIT-pthreads on some platforms, this means you need to

use `-h` or `--host` with a value other than `localhost` when you run client programs.

- When MySQL is compiled using MIT-pthreads, system locking is disabled by default for performance reasons. You can tell the server to use system locking with the `--external-locking` option. This is needed only if you want to be able to run two MySQL servers against the same data files, but that is not recommended, anyway.
- Sometimes the `pthread bind()` command fails to bind to a socket without any error message (at least on Solaris). The result is that all connections to the server fail. For example:

```
shell> mysqladmin version
mysqladmin: connect to server at '' failed;
error: 'Can't connect to mysql server on localhost (146)'
```

The solution to this problem is to kill the **mysqld** server and restart it. This has happened to us only when we have forcibly stopped the server and restarted it immediately.

- With MIT-pthreads, the `sleep()` system call isn't interruptible with `SIGINT` (break). This is noticeable only when you run **mysqladmin --sleep**. You must wait for the `sleep()` call to terminate before the interrupt is served and the process stops.
- When linking, you might receive warning messages like these (at least on Solaris); they can be ignored:

```
ld: warning: symbol `__iob' has differing sizes:
  (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x
file /usr/lib/libc.so value=0x140);
  /my/local/pthreads/lib/libpthread.a(findfp.o) definition tak
ld: warning: symbol `__iob' has differing sizes:
  (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x
file /usr/lib/libc.so value=0x140);
  /my/local/pthreads/lib/libpthread.a(findfp.o) definition tak
```

- Some other warnings also can be ignored:

```
implicit declaration of function `int strtoll(...)'
implicit declaration of function `int strtoul(...)'
```

- We have not been able to make `readline` work with MIT-pthreads. (This is not necessary, but may be of interest to some.)

2.9.6. Installing MySQL from Source on Windows

These instructions describe how to build binaries from source for MySQL 5.0 on Windows. Instructions are provided for building binaries from a standard source distribution or from the BitKeeper tree that contains the latest development source.

Note: The instructions here are strictly for users who want to test MySQL on Windows from the latest source distribution or from the BitKeeper tree. For production use, MySQL AB does not advise using a MySQL server built by yourself from source. Normally, it is best to use precompiled binary distributions of MySQL that are built specifically for optimal performance on Windows by MySQL AB. Instructions for installing a binary distributions are available in [Section 2.3, “Installing MySQL on Windows”](#).

To build MySQL on Windows from source, you need the following compiler and resources available on your Windows system:

- Visual Studio .Net 2003 (7.1) compiler system
- Between 3GB and 5GB disk space.
- Windows XP, Windows 2000 or higher.

The exact system requirements can be found here:

<http://msdn.microsoft.com/vstudio/Previous/2003/sysreqs/default.aspx>

You also need a MySQL source distribution for Windows. There are two ways to obtain a source distribution:

1. Obtain a Windows source distribution packaged by MySQL AB for the particular version of MySQL in which you are interested. These are available from <http://dev.mysql.com/downloads/>.
2. You can package a source distribution yourself from the latest BitKeeper developer source tree. If you plan to do this, you must create the package on a Unix system and then transfer it to your Windows system. (Some of the

configuration and build steps require tools that work only on Unix.) The BitKeeper approach thus requires:

- A system running Unix, or a Unix-like system such as Linux.
- BitKeeper installed on that system. See [Section 2.9.3, “Installing from the Development Source Tree”](#), for instructions how to download and install BitKeeper.

If you are using a Windows source distribution, you can go directly to [Section 2.9.6.1, “Building MySQL Using VC++”](#). To build from the BitKeeper tree, proceed to [Section 2.9.6.2, “Creating a Windows Source Package from the Latest Development Source”](#).

If you find something not working as expected, or you have suggestions about ways to improve the current build process on Windows, please send a message to the win32 mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#).

2.9.6.1. Building MySQL Using VC++

Note: VC++ workspace files for MySQL 4.1 and above are compatible with Microsoft Visual Studio 7.1 and tested by MySQL AB staff before each release.

Follow this procedure to build MySQL:

1. Create a work directory (for example, C:\workdir).
2. Unpack the source distribution in the aforementioned directory using **WinZip** or another Windows tool that can read .zip files.
3. Start Visual Studio .Net 2003 (7.1).
4. From the File menu, select Open Solution....
5. Open the mysql.sln solution you find in the work directory.
6. From the Build menu, select Configuration Manager....
7. In the Active Solution Configuration pop-up menu, select the configuration to use. You likely want to use one of nt (normal server, not for Windows

98/ME), Max nt (more engines and features, not for 98/ME) or Debug configuration.

8. From the Build menu, select Build Solution.
9. Debug versions of the programs and libraries are placed in the `client_debug` and `lib_debug` directories. Release versions of the programs and libraries are placed in the `client_release` and `lib_release` directories.
10. Test the server. The server built using the preceding instructions expects that the MySQL base directory and data directory are `C:\mysql` and `C:\mysql\data` by default. If you want to test your server using the source tree root directory and its data directory as the base directory and data directory, you need to tell the server their pathnames. You can either do this on the command line with the `--basedir` and `--datadir` options, or by placing appropriate options in an option file. (See [Section 4.3.2, “Using Option Files”](#).) If you have an existing data directory elsewhere that you want to use, you can specify its pathname instead.
11. Start your server from the `client_release` or `client_debug` directory, depending on which server you built or want to use. The general server startup instructions are in [Section 2.3, “Installing MySQL on Windows”](#). You must adapt the instructions appropriately if you want to use a different base directory or data directory.
12. When the server is running in standalone fashion or as a service based on your configuration, try to connect to it from the **mysql** interactive command-line utility that exists in your `client_release` or `client_debug` directory.

When you are satisfied that the programs you have built are working correctly, stop the server. Then install MySQL as follows:

1. Create the directories where you want to install MySQL. For example, to install into `C:\mysql`, use these commands:

```
C:\> mkdir C:\mysql
C:\> mkdir C:\mysql\bin
C:\> mkdir C:\mysql\data
C:\> mkdir C:\mysql\share
```

```
C:\> mkdir C:\mysql\scripts
```

If you want to compile other clients and link them to MySQL, you should also create several additional directories:

```
C:\> mkdir C:\mysql\include
C:\> mkdir C:\mysql\lib
C:\> mkdir C:\mysql\lib\debug
C:\> mkdir C:\mysql\lib\opt
```

If you want to benchmark MySQL, create this directory:

```
C:\> mkdir C:\mysql\sql-bench
```

Benchmarking requires Perl support. See [Section 2.14, “Perl Installation Notes”](#).

2. From the `workdir` directory, copy into the `C:\mysql` directory the following directories:

```
C:\> cd \workdir
C:\workdir> copy client_release\*.exe C:\mysql\bin
C:\workdir> copy client_debug\mysqld.exe C:\mysql\bin\mysqld-det
C:\workdir> xcopy scripts\*. * C:\mysql\scripts /E
C:\workdir> xcopy share\*. * C:\mysql\share /E
```

If you want to compile other clients and link them to MySQL, you should also copy several libraries and header files:

```
C:\workdir> copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
C:\workdir> copy lib_debug\libmysql.* C:\mysql\lib\debug
C:\workdir> copy lib_debug\zlib.* C:\mysql\lib\debug
C:\workdir> copy lib_release\mysqlclient.lib C:\mysql\lib\opt
C:\workdir> copy lib_release\libmysql.* C:\mysql\lib\opt
C:\workdir> copy lib_release\zlib.* C:\mysql\lib\opt
C:\workdir> copy include\*.h C:\mysql\include
C:\workdir> copy libmysql\libmysql.def C:\mysql\include
```

If you want to benchmark MySQL, you should also do this:

```
C:\workdir> xcopy sql-bench\*. * C:\mysql\bench /E
```

Set up and start the server in the same way as for the binary Windows distribution. See [Section 2.3, “Installing MySQL on Windows”](#).

2.9.6.2. Creating a Windows Source Package from the Latest Development Source

To create a Windows source package from the current BitKeeper source tree, use the instructions here. This procedure must be performed on a system running a Unix or Unix-like operating system because some of the configuration and build steps require tools that work only on Unix. For example, the following procedure is known to work well on Linux.

1. Copy the BitKeeper source tree for MySQL 5.0. For instructions on how to do this, see [Section 2.9.3, “Installing from the Development Source Tree”](#).
2. Configure and build the distribution so that you have a server binary to work with. One way to do this is to run the following command in the top-level directory of your source tree:

```
shell> ./BUILD/compile-pentium-max
```

3. After making sure that the build process completed successfully, run the following utility script from top-level directory of your source tree:

```
shell> ./scripts/make_win_src_distribution
```

This script creates a Windows source package to be used on your Windows system. You can supply different options to the script based on your needs. It accepts the following options:

- --help

Display a help message.

- --debug

Print information about script operations, do not create package.

- --tmp

Specify the temporary location.

- --suffix

The suffix name for the package.

- `--dirname`

Directory name to copy files (intermediate).

- `--silent`

Do not print verbose list of files processed.

- `--tar`

Create `tar.gz` package instead of `.zip` package.

By default, **`make_win_src_distribution`** creates a Zip-format archive with the name `mysql-VERSION-win-src.zip`, where *VERSION* represents the version of your MySQL source tree.

4. Copy or upload the Windows source package that you have just created to your Windows machine. To compile it, use the instructions in [Section 2.9.6.1, “Building MySQL Using VC++”](#).

2.9.7. Compiling MySQL Clients on Windows

In your source files, you should include `my_global.h` before `mysql.h`:

```
#include <my_global.h>
#include <mysql.h>
```

`my_global.h` includes any other files needed for Windows compatibility (such as `windows.h`) if you compile your program on Windows.

You can either link your code with the dynamic `libmysql.lib` library, which is just a wrapper to load in `libmysql.dll` on demand, or link with the static `mysqlclient.lib` library.

The MySQL client libraries are compiled as threaded libraries, so you should also compile your code to be multi-threaded.

2.10. Post-Installation Setup and Testing

After installing MySQL, there are some issues that you should address. For example, on Unix, you should initialize the data directory and create the MySQL grant tables. On all platforms, an important security concern is that the initial accounts in the grant tables have no passwords. You should assign passwords to prevent unauthorized access to the MySQL server. Optionally, you can create time zone tables to enable recognition of named time zones.

The following sections include post-installation procedures that are specific to Windows systems and to Unix systems. Another section, [Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”](#), applies to all platforms; it describes what to do if you have trouble getting the server to start. [Section 2.10.3, “Securing the Initial MySQL Accounts”](#), also applies to all platforms. You should follow its instructions to make sure that you have properly protected your MySQL accounts by assigning passwords to them.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in [Section 5.8, “The MySQL Access Privilege System”](#), and [Section 5.9, “MySQL User Account Management”](#).

2.10.1. Windows Post-Installation Procedures

On Windows, the data directory and the grant tables do not have to be created. MySQL Windows distributions include the grant tables with a set of preinitialized accounts in the `mysql` database under the data directory. It is unnecessary to run the `mysql_install_db` script that is used on Unix. Regarding passwords, if you installed MySQL using the Windows Installation Wizard, you may have already assigned passwords to the accounts. (See [Section 2.3.3, “Using the MySQL Installation Wizard”](#).) Otherwise, use the password-assignment procedure given in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

Before setting up passwords, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure that the server is running (see [Section 2.3.9, “Starting the](#)

[Server for the First Time](#)”), and then issue the following commands to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```
C:\> C:\mysql\bin\mysqlshow
```

```
+-----+
| Databases |
+-----+
| mysql     |
| test      |
+-----+
```

```
C:\> C:\mysql\bin\mysqlshow mysql
```

```
Database: mysql
```

```
+-----+
|          Tables          |
+-----+
| columns_priv             |
| db                       |
| func                    |
| help_category           |
| help_keyword            |
| help_relation           |
| help_topic              |
| host                    |
| proc                   |
| procs_priv              |
| tables_priv             |
| time_zone               |
| time_zone_leap_second   |
| time_zone_name          |
| time_zone_transition    |
| time_zone_transition_type |
| user                    |
+-----+
```

```
C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM db" mysql
```

```
+-----+-----+-----+
| host | db   | user |
+-----+-----+-----+
| %    | test% |     |
+-----+-----+-----+
```

If you are running a version of Windows that supports services and you want the MySQL server to run automatically when Windows starts, see [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

2.10.2. Unix Post-Installation Procedures

After installing MySQL on Unix, you need to initialize the grant tables, start the server, and make sure that the server works satisfactorily. You may also wish to arrange for the server to be started and stopped automatically when your system starts and stops. You should also assign passwords to the accounts in the grant tables.

On Unix, the grant tables are set up by the **mysql_install_db** program. For some installation methods, this program is run for you automatically:

- If you install MySQL on Linux using RPM distributions, the server RPM runs **mysql_install_db**.
- If you install MySQL on Mac OS X using a PKG distribution, the installer runs **mysql_install_db**.

Otherwise, you will need to run **mysql_install_db** yourself.

The following procedure describes how to initialize the grant tables (if that has not previously been done) and then start the server. It also suggests some commands that you can use to test whether the server is accessible and working properly. For information about starting and stopping the server automatically, see [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).

After you complete the procedure and have the server running, you should assign passwords to the accounts created by **mysql_install_db**. Instructions for doing so are given in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

In the examples shown here, the server runs under the user ID of the `mysql` login account. This assumes that such an account exists. Either create the account if it does not exist, or substitute the name of a different existing login account that you plan to use for running the server.

1. Change location into the top-level directory of your MySQL installation, represented here by *BASEDIR*:

```
shell> cd BASEDIR
```

BASEDIR is likely to be something like `/usr/local/mysql` or `/usr/local`.

The following steps assume that you are located in this directory.

2. If necessary, run the **mysql_install_db** program to set up the initial MySQL grant tables containing the privileges that determine how users are allowed to connect to the server. You'll need to do this if you used a distribution type for which the installation procedure doesn't run the program for you.

Typically, **mysql_install_db** needs to be run only the first time you install MySQL, so you can skip this step if you are upgrading an existing installation. However, **mysql_install_db** does not overwrite any existing privilege tables, so it should be safe to run in any circumstances.

To initialize the grant tables, use one of the following commands, depending on whether **mysql_install_db** is located in the `bin` or `scripts` directory:

```
shell> bin/mysql_install_db --user=mysql
shell> scripts/mysql_install_db --user=mysql
```

The **mysql_install_db** script creates the server's data directory. Under the data directory, it creates directories for the `mysql` database that holds all database privileges and the `test` database that you can use to test MySQL. The script also creates privilege table entries for `root` and `anonymous-user` accounts. The accounts have no passwords initially. A description of their initial privileges is given in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#). Briefly, these privileges allow the MySQL `root` user to do anything, and allow anybody to create or use databases with a name of `test` or starting with `test_`.

It is important to make sure that the database directories and files are owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this, the `--user` option should be used as shown if you run **mysql_install_db** as `root`. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` option from the command.

mysql_install_db creates several tables in the `mysql` database, including `user`, `db`, `host`, `tables_priv`, `columns_priv`, `func`, and others. See [Section 5.8, “The MySQL Access Privilege System”](#), for a complete listing and description of these tables.

If you don't want to have the test database, you can remove it with **mysqladmin -u root drop test** after starting the server.

If you have trouble with **mysql_install_db** at this point, see [Section 2.10.2.1, “Problems Running mysql_install_db”](#).

3. Start the MySQL server:

```
shell> bin/mysqld_safe --user=mysql &
```

It is important that the MySQL server be run using an unprivileged (non-root) login account. To ensure this, the `--user` option should be used as shown if you run `mysql_safe` as system root. Otherwise, you should execute the script while logged in to the system as `mysql`, in which case you can omit the `--user` option from the command.

Further instructions for running MySQL as an unprivileged user are given in [Section 5.7.5, “How to Run MySQL as a Normal User”](#).

If you neglected to create the grant tables before proceeding to this step, the following message appears in the error log file when you start the server:

```
mysqld: Can't find file: 'host.frm'
```

If you have other problems starting the server, see [Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”](#).

4. Use **mysqladmin** to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

The output from **mysqladmin version** varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell> bin/mysqladmin version
mysqladmin Ver 14.12 Distrib 5.0.25, for pc-linux-gnu on i686
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult
This software comes with ABSOLUTELY NO WARRANTY. This is free software
and you are welcome to modify and redistribute it under the GPL
```

```
Server version          5.0.25-Max
Protocol version       10
Connection             Localhost via UNIX socket
UNIX socket            /var/lib/mysql/mysql.sock
Uptime:                14 days 5 hours 5 min 21 sec
```

```
Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
Queries per second avg: 0.000
```

To see what else you can do with **mysqladmin**, invoke it with the `--help` option.

5. Verify that you can shut down the server:

```
shell> bin/mysqladmin -u root shutdown
```

6. Verify that you can start the server again. Do this by using **mysqld_safe** or by invoking **mysqld** directly. For example:

```
shell> bin/mysqld_safe --user=mysql --log &
```

If **mysqld_safe** fails, see [Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”](#).

7. Run some simple tests to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```
shell> bin/mysqlshow
```

```
+-----+
| Databases |
+-----+
| mysql     |
| test      |
+-----+
```

```
shell> bin/mysqlshow mysql
```

```
Database: mysql
+-----+
|          Tables          |
+-----+
| columns_priv            |
| db                      |
| func                   |
| help_category          |
```

```

| help_keyword          |
| help_relation        |
| help_topic            |
| host                  |
| proc                  |
| procs_priv            |
| tables_priv           |
| time_zone             |
| time_zone_leap_second |
| time_zone_name        |
| time_zone_transition  |
| time_zone_transition_type |
| user                  |
+-----+-----+-----+

```

```
shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
```

```

+-----+-----+-----+
| host | db      | user |
+-----+-----+-----+
| %    | test   |      |
| %    | test_% |      |
+-----+-----+-----+

```

8. There is a benchmark suite in the `sql-bench` directory (under the MySQL installation directory) that you can use to compare how MySQL performs on different platforms. The benchmark suite is written in Perl. It requires the Perl DBI module that provides a database-independent interface to the various databases, and some other additional Perl modules:

```

DBI
DBD::mysql
Data::Dumper
Data::ShowTable

```

These modules can be obtained from CPAN (<http://www.cpan.org/>). See also [Section 2.14.1, “Installing Perl on Unix”](#).

The `sql-bench/Results` directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

```

shell> cd sql-bench
shell> perl run-all-tests

```

If you don't have the `sql-bench` directory, you probably installed MySQL using RPM files other than the source RPM. (The source RPM includes the

mysql-bench benchmark directory.) In this case, you must first install the benchmark suite before you can use it. There are separate benchmark RPM files named `mysql-bench-VERSION-i386.rpm` that contain benchmark code and data.

If you have a source distribution, there are also tests in its `tests` subdirectory that you can run. For example, to run `auto_increment.tst`, execute this command from the top-level directory of your source distribution:

```
shell> mysql -vfv test < ./tests/auto_increment.tst
```

The expected result of the test can be found in the `./tests/auto_increment.res` file.

9. At this point, you should have the server running. However, none of the initial MySQL accounts have a password, so you should assign passwords using the instructions found in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

The MySQL 5.0 installation procedure creates time zone tables in the `mysql` database. However, you must populate the tables manually using the instructions in [Section 5.11.8, “MySQL Server Time Zone Support”](#).

2.10.2.1. Problems Running `mysql_install_db`

The purpose of the `mysql_install_db` script is to generate new MySQL privilege tables. It does not overwrite existing MySQL privilege tables, and it does not affect any other data.

If you want to re-create your privilege tables, first stop the `mysqld` server if it's running. Then rename the `mysql` directory under the data directory to save it, and then run `mysql_install_db`. Suppose that your current directory is the MySQL installation directory and that `mysql_install_db` is located in the `bin` directory and the data directory is named `data`. To rename the `mysql` database and re-run `mysql_install_db`, use these commands.

```
shell> mv data/mysql data/mysql.old
shell> bin/mysql_install_db --user=mysql
```

When you run **mysql_install_db**, you might encounter the following problems:

- **mysql_install_db** fails to install the grant tables

You may find that **mysql_install_db** fails to install the grant tables and terminates after displaying the following messages:

```
Starting mysqld daemon with databases from XXXXXX  
mysqld ended
```

In this case, you should examine the error log file very carefully. The log should be located in the directory XXXXXX named by the error message and should indicate why **mysqld** didn't start. If you do not understand what happened, include the log when you post a bug report. See [Section 1.8, “How to Report Bugs or Problems”](#).

- **There is a mysqld** process running

This indicates that the server is running, in which case the grant tables have probably been created already. If so, there is no need to run **mysql_install_db** at all because it needs to be run only once (when you install MySQL the first time).

- **Installing a second mysqld** server does not work when one server is running

This can happen when you have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation, but you want to create a second installation for testing purposes. Generally the problem that occurs when you try to run a second server is that it tries to use a network interface that is in use by the first server. In this case, you should see one of the following error messages:

```
Can't start server: Bind on TCP/IP port:  
Address already in use  
Can't start server: Bind on unix socket...
```

For instructions on setting up multiple servers, see [Section 5.13, “Running Multiple MySQL Servers on the Same Machine”](#).

- **You do not have write access to the /tmp directory**

If you do not have write access to create temporary files or a Unix socket file in the default location (the /tmp directory), an error occurs when you run **mysql_install_db** or the **mysqld** server.

You can specify different locations for the temporary directory and Unix socket file by executing these commands prior to starting **mysql_install_db** or **mysqld**, where *some_tmp_dir* is the full pathname to some directory for which you have write permission:

```
shell> TMPDIR=/some_tmp_dir/  
shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock  
shell> export TMPDIR MYSQL_UNIX_PORT
```

Then you should be able to run **mysql_install_db** and start the server with these commands:

```
shell> bin/mysql_install_db --user=mysql  
shell> bin/mysqld_safe --user=mysql &
```

If **mysql_install_db** is located in the scripts directory, modify the first command to `scripts/mysql_install_db`.

See [Section A.4.5, “How to Protect or Change the MySQL Unix Socket File”](#), and [Appendix F, *Environment Variables*](#).

There are some alternatives to running the **mysql_install_db** script provided in the MySQL distribution:

- If you want the initial privileges to be different from the standard defaults, you can modify **mysql_install_db** before you run it. However, it is preferable to use GRANT and REVOKE to change the privileges *after* the grant tables have been set up. In other words, you can run **mysql_install_db**, and then use `mysql -u root mysql` to connect to the server as the MySQL root user so that you can issue the necessary GRANT and REVOKE statements.

If you want to install MySQL on several machines with the same privileges, you can put the GRANT and REVOKE statements in a file and execute the file as a script using `mysql` after running **mysql_install_db**. For example:

```
shell> bin/mysql_install_db --user=mysql
shell> bin/mysql -u root < your_script_file
```

By doing this, you can avoid having to issue the statements manually on each machine.

- It is possible to re-create the grant tables completely after they have previously been created. You might want to do this if you're just learning how to use GRANT and REVOKE and have made so many modifications after running **mysql_install_db** that you want to wipe out the tables and start over.

To re-create the grant tables, remove all the .frm, .MYI, and .MYD files in the mysql database directory. Then run the **mysql_install_db** script again.

- You can start **mysqld** manually using the `--skip-grant-tables` option and add the privilege information yourself using **mysql**:

```
shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
shell> bin/mysql mysql
```

From **mysql**, manually execute the SQL commands contained in **mysql_install_db**. Make sure that you run **mysqladmin flush-privileges** or **mysqladmin reload** afterward to tell the server to reload the grant tables.

Note that by not using **mysql_install_db**, you not only have to populate the grant tables manually, you also have to create them first.

2.10.2.2. Starting and Stopping MySQL Automatically

Generally, you start the **mysqld** server in one of these ways:

- By invoking **mysqld** directly. This works on any platform.
- By running the MySQL server as a Windows service. This can be done on versions of Windows that support services (such as NT, 2000, XP, and 2003). The service can be set to start the server automatically when Windows starts, or as a manual service that you start on request. For instructions, see [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

- By invoking **mysqld_safe**, which tries to determine the proper options for **mysqld** and then runs it with those options. This script is used on Unix and Unix-like systems. See [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#).
- By invoking **mysql.server**. This script is used primarily at system startup and shutdown on systems that use System V-style run directories, where it usually is installed under the name `mysql`. The **mysql.server** script starts the server by invoking **mysqld_safe**. See [Section 5.4.2, “mysql.server — MySQL Server Startup Script”](#).
- On Mac OS X, you can install a separate MySQL Startup Item package to enable the automatic startup of MySQL on system startup. The Startup Item starts the server by invoking **mysql.server**. See [Section 2.5, “Installing MySQL on Mac OS X”](#), for details.

The **mysqld_safe** and **mysql.server** scripts and the Mac OS X Startup Item can be used to start the server manually, or automatically at system startup time. **mysql.server** and the Startup Item also can be used to stop the server.

To start or stop the server manually using the **mysql.server** script, invoke it with `start` or `stop` arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

Before **mysql.server** starts the server, it changes location to the MySQL installation directory, and then invokes **mysqld_safe**. If you want the server to run as some specific user, add an appropriate user option to the `[mysqld]` group of the `/etc/my.cnf` option file, as shown later in this section. (It is possible that you will need to edit **mysql.server** if you've installed a binary distribution of MySQL in a non-standard location. Modify it to `cd` into the proper directory before it runs **mysqld_safe**. If you do this, your modified version of **mysql.server** may be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.)

mysql.server stop stops the server by sending a signal to it. You can also stop the server manually by executing **mysqladmin shutdown**.

To start and stop MySQL automatically on your server, you need to add `start` and

stop commands to the appropriate places in your `/etc/rc*` files.

If you use the Linux server RPM package (`MySQL-server-VERSION.rpm`), the **mysql.server** script is installed in the `/etc/init.d` directory with the name `mysql`. You need not install it manually. See [Section 2.4, “Installing MySQL on Linux”](#), for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as **mysqld**.

If you install MySQL from a source distribution or using a binary distribution format that does not install **mysql.server** automatically, you can install it manually. The script can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree.

To install **mysql.server** manually, copy it to the `/etc/init.d` directory with the name **mysql**, and then make it executable. Do this by changing location into the appropriate directory where **mysql.server** is located and executing these commands:

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

Older Red Hat systems use the `/etc/rc.d/init.d` directory rather than `/etc/init.d`. Adjust the preceding commands accordingly. Alternatively, first create `/etc/init.d` as a symbolic link that points to `/etc/rc.d/init.d`:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use `chkconfig`:

```
shell> chkconfig --add mysql
```

On some Linux systems, the following command also seems to be necessary to fully enable the **mysql** script:

```
shell> chkconfig --level 345 mysql on
```

On FreeBSD, startup scripts generally should go in `/usr/local/etc/rc.d/`. The `rc(8)` manual page states that scripts in this directory are executed only if their

basename matches the *.sh shell filename pattern. Any other files or directories present within the directory are silently ignored. In other words, on FreeBSD, you should install the `mysql.server` script as `/usr/local/etc/rc.d/mysql.server.sh` to enable automatic startup.

As an alternative to the preceding setup, some operating systems also use `/etc/rc.local` or `/etc/init.d/boot.local` to start additional services on startup. To start up MySQL using this method, you could append a command like the one following to the appropriate startup file:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

For other systems, consult your operating system documentation to see how to install startup scripts.

You can add options for **mysql.server** in a global `/etc/my.cnf` file. A typical `/etc/my.cnf` file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql
```

```
[mysql.server]
basedir=/usr/local/mysql
```

The **mysql.server** script understands the following options: `basedir`, `datadir`, and `pid-file`. If specified, they *must* be placed in an option file, not on the command line. **mysql.server** understands only `start` and `stop` as command-line arguments.

The following table shows which option groups the server and each startup script read from option files:

Script	Option Groups
mysqld	[mysqld], [server], [mysqld-major_version]
mysqld_safe	[mysqld], [server], [mysqld_safe]
mysql.server	[mysqld], [mysql.server], [server]

[mysqld-major_version] means that groups with names like [mysqld-4.1] and

[mysqld-5.0] are read by servers having versions 4.1.x, 5.0.x, and so forth. This feature can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, **mysql.server** also reads the [mysql_server] group and **mysqld_safe** also reads the [safe_mysqld] group. However, you should update your option files to use the [mysql.server] and [mysqld_safe] groups instead when using MySQL 5.0.

See [Section 4.3.2, “Using Option Files”](#).

2.10.2.3. Starting and Troubleshooting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server on Unix. If you are using Windows, see [Section 2.3.13, “Troubleshooting a MySQL Installation Under Windows”](#).

If you have problems starting the server, here are some things to try:

- Check the error log to see why the server does not start.
- Specify any special options needed by the storage engines you are using.
- Make sure that the server knows where to find the data directory.
- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must be set such that the server can read and modify them.
- Verify that the network interfaces the server wants to use are available.

Some storage engines have options that control their behavior. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables (InnoDB, BDB, NDB), be sure that you have them configured the way you want before starting the server:

- If you are using InnoDB tables, see [Section 14.2.3, “InnoDB Configuration”](#).
- If you are using BDB (Berkeley DB) tables, see [Section 14.5.3, “BDB Startup”](#).

[Options](#)".

- If you are using MySQL Cluster, see [Section 15.4, "MySQL Cluster Configuration"](#).

Storage engines will use default option values if you specify none, but it is recommended that you review the available options and specify explicit values for those for which the defaults are not appropriate for your installation.

When the **mysqld** server starts, it changes location to the data directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

The data directory location is hardwired in when the server is compiled. This is where the server looks for the data directory by default. If the data directory is located somewhere else on your system, the server will not work properly. You can determine what the default path settings are by invoking **mysqld** with the `--verbose` and `--help` options.

If the default locations don't match the MySQL installation layout on your system, you can override them by specifying options to **mysqld** or **mysqld_safe** on the command line or in an option file.

To specify the location of the data directory explicitly, use the `--datadir` option. However, normally you can tell **mysqld** the location of the base directory under which MySQL is installed and it looks for the data directory there. You can do this with the `--basedir` option.

To check the effect of specifying path options, invoke **mysqld** with those options followed by the `--verbose` and `--help` options. For example, if you change location into the directory where **mysqld** is installed and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

You can specify other options such as `--datadir` as well, but `--verbose` and `--help` must be the last options.

Once you determine the path settings you want, start the server without `--`

verbose and `--help`.

If **mysqld** is currently running, you can find out what path settings it is using by executing this command:

```
shell> mysqladmin variables
```

Or:

```
shell> mysqladmin -h host_name variables
```

host_name is the name of the MySQL server host.

If you get Errcode 13 (which means Permission denied) when starting **mysqld**, this means that the privileges of the data directory or its contents do not allow the server access. In this case, you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as root, but this raises security issues and should be avoided.

On Unix, change location into the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

```
shell> ls -la /usr/local/mysql/var
```

If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named `mysql`, use these commands:

```
shell> chown -R mysql /usr/local/mysql/var  
shell> chgrp -R mysql /usr/local/mysql/var
```

If the server fails to start up correctly, check the error log. Log files are located in the data directory (typically `C:\Program Files\MySQL\MySQL Server 5.0\data` on Windows, `/usr/local/mysql/data` for a Unix binary distribution, and `/usr/local/var` for a Unix source distribution). Look in the data directory for files with names of the form `host_name.err` and `host_name.log`, where *host_name* is the name of your server host. Then examine the last few lines of these files. On Unix, you can use `tail` to display them:

```
shell> tail host_name.err  
shell> tail host_name.log
```

The error log should contain information that indicates why the server couldn't start. For example, you might see something like this in the log:

```
000729 14:50:10 bdb: Recovery function for LSN 1 27595 failed
000729 14:50:10 bdb: warning: ./test/t1.db: No such file or direct
000729 14:50:10 Can't init databases
```

This means that you did not start **mysqld** with the `--bdb-no-recover` option and Berkeley DB found something wrong with its own log files when it tried to recover your databases. To be able to continue, you should move the old Berkeley DB log files from the database directory to some other place, where you can later examine them. The BDB log files are named in sequence beginning with `log.0000000001`, where the number increases over time.

If you are running **mysqld** with BDB table support and **mysqld** dumps core at startup, this could be due to problems with the BDB recovery log. In this case, you can try starting **mysqld** with `--bdb-no-recover`. If that helps, you should remove all BDB log files from the data directory and try starting **mysqld** again without the `--bdb-no-recover` option.

If either of the following errors occur, it means that some other program (perhaps another **mysqld** server) is using the TCP/IP port or Unix socket file that **mysqld** is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

Use **ps** to determine whether you have another **mysqld** server running. If so, shut down the server before starting **mysqld** again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in [Section 5.13, “Running Multiple MySQL Servers on the Same Machine”](#).)

If no other server is running, try to execute the command `telnet your_host_name tcp_ip_port_number`. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you don't get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that **mysqld** is trying to use. You'll need to track down what program this is and disable it, or else tell **mysqld** to listen to a different port with the `--port` option. In this case, you'll also need to specify the port number for client programs when connecting to the server via TCP/IP.

Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to allow access to the port.

If the server starts but you can't connect to it, you should make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1      localhost
```

This problem occurs only on systems that do not have a working thread library and for which MySQL must be configured to use MIT-pthreads.

If you cannot get **mysqld** to start, you can try to make a trace file to find the problem by using the `--debug` option. See [Section E.1.2, “Creating Trace Files”](#).

2.10.3. Securing the Initial MySQL Accounts

Part of the MySQL installation process is to set up the `mysql` database that contains the grant tables:

- Windows distributions contain preinitialized grant tables that are installed automatically.
- On Unix, the grant tables are populated by the **mysql_install_db** program. Some installation methods run this program for you. Others require that you execute it manually. For details, see [Section 2.10.2, “Unix Post-Installation Procedures”](#).

The grant tables define the initial MySQL user accounts and their access privileges. These accounts are set up as follows:

- Accounts with the username `root` are created. These are superuser accounts that can do anything. The initial `root` account passwords are empty, so anyone can connect to the MySQL server as `root` — *without a password* — and be granted all privileges.
 - On Windows, one `root` account is created; this account allows connecting from the local host only. The Windows installer will optionally create an account allowing for connections from any host only if the user selects the Enable root access from remote machines

option during installation.

- On Unix, both root accounts are for connections from the local host. Connections must be made from the local host by specifying a hostname of `localhost` for one of the accounts, or the actual hostname or IP number for the other.
- Two anonymous-user accounts are created, each with an empty username. The anonymous accounts have no password, so anyone can use them to connect to the MySQL server.
 - On Windows, one anonymous account is for connections from the local host. It has all privileges, just like the root accounts. The other is for connections from any host and has all privileges for the test database and for other databases with names that start with test.
 - On Unix, both anonymous accounts are for connections from the local host. Connections must be made from the local host by specifying a hostname of `localhost` for one of the accounts, or the actual hostname or IP number for the other. These accounts have all privileges for the test database and for other databases with names that start with `test_`.

As noted, none of the initial accounts have passwords. This means that your MySQL installation is unprotected until you do something about it:

- If you want to prevent clients from connecting as anonymous users without a password, you should either assign a password to each anonymous account or else remove the accounts.
- You should assign a password to each MySQL root account.

The following instructions describe how to set up passwords for the initial MySQL accounts, first for the anonymous accounts and then for the root accounts. Replace “*newpwd*” in the examples with the actual password that you want to use. The instructions also cover how to remove the anonymous accounts, should you prefer not to allow anonymous access at all.

You might want to defer setting the passwords until later, so that you don't need to specify them while you perform additional setup or testing. However, be sure

to set them before using your installation for production purposes.

Anonymous Account Password Assignment

To assign passwords to the anonymous accounts, connect to the server as root and then use either SET PASSWORD or UPDATE. In either case, be sure to encrypt the password using the PASSWORD() function.

To use SET PASSWORD on Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR '@localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR '@%' = PASSWORD('newpwd');
```

To use SET PASSWORD on Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR '@localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR '@host_name' = PASSWORD('newpwd');
```

In the second SET PASSWORD statement, replace *host_name* with the name of the server host. This is the name that is specified in the Host column of the non-localhost record for root in the user table. If you don't know what hostname this is, issue the following statement before using SET PASSWORD:

```
mysql> SELECT Host, User FROM mysql.user;
```

Look for the record that has root in the User column and something other than localhost in the Host column. Then use that Host value in the second SET PASSWORD statement.

The other way to assign passwords to the anonymous accounts is by using UPDATE to modify the user table directly. Connect to the server as root and issue an UPDATE statement that assigns a value to the Password column of the appropriate user table records. The procedure is the same for Windows and Unix. The following UPDATE statement assigns a password to both anonymous accounts at once:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
-> WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

After you update the passwords in the user table directly using UPDATE, you must tell the server to re-read the grant tables with FLUSH PRIVILEGES. Otherwise, the change goes unnoticed until you restart the server.

Anonymous Account Removal

If you prefer to remove the anonymous accounts instead, do so as follows:

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

The DELETE statement applies both to Windows and to Unix. On Windows, if you want to remove only the anonymous account that has the same privileges as root, do this instead:

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
```

That account allows anonymous access but has full privileges, so removing it improves security.

root Account Password Assignment

You can assign passwords to the root accounts in several ways. The following discussion demonstrates three methods:

- Use the SET PASSWORD statement
- Use the **mysqladmin** command-line client program
- Use the UPDATE statement

To assign passwords using SET PASSWORD, connect to the server as root and issue two SET PASSWORD statements. Be sure to encrypt the password using the PASSWORD() function.

For Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
```

```
mysql> SET PASSWORD FOR 'root'@ '%' = PASSWORD('newpwd');
```

For Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

In the second SET PASSWORD statement, replace *host_name* with the name of the server host. This is the same hostname that you used when you assigned the anonymous account passwords.

To assign passwords to the root accounts using **mysqladmin**, execute the following commands:

```
shell> mysqladmin -u root password "newpwd"
shell> mysqladmin -u root -h host_name password "newpwd"
```

These commands apply both to Windows and to Unix. In the second command, replace *host_name* with the name of the server host. The double quotes around the password are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter.

You can also use UPDATE to modify the user table directly. The following UPDATE statement assigns a password to both root accounts at once:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
-> WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

The UPDATE statement applies both to Windows and to Unix.

After the passwords have been set, you must supply the appropriate password whenever you connect to the server. For example, if you want to use **mysqladmin** to shut down the server, you can do so using this command:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

Note: If you forget your root password after setting it up, [Section A.4.1, “How to Reset the Root Password”](#), covers the procedure for resetting it.

To set up additional accounts, you can use the GRANT statement. For instructions, see [Section 5.9.2, “Adding New User Accounts to MySQL”](#).

2.11. Upgrading MySQL

As a general rule, we recommend that when upgrading from one release series to another, you should go to the next series rather than skipping a series. For example, if you currently are running MySQL 3.23 and wish to upgrade to a newer series, upgrade to MySQL 4.0 rather than to 4.1 or 5.0.

The following items form a checklist of things that you should do whenever you perform an upgrade:

- Before upgrading from MySQL 4.1 to 5.0, read [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#)) as well as [Appendix D, *MySQL Change History*](#). These provide information about features that are new in MySQL 5.0 or differ from those found in MySQL 4.1. If you wish to upgrade from a release series previous to MySQL 4.1, you should upgrade to each successive release series in turn until you have reached MySQL 4.1, and then proceed with the upgrade to MySQL 5.0. For information on upgrading from MySQL 4.1 or earlier releases, see the *MySQL 3.23, 4.0, 4.1 Reference Manual*.
- Before you perform an upgrade, back up your databases, including the `mysql` database that contains the grant tables.
- Some releases of MySQL introduce incompatible changes to tables. (Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.) Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.

To avoid problems due to such changes, after you upgrade to a new version of MySQL, you should run **mysql_upgrade** to check your tables (and repair them if necessary), and to update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

- If you are running MySQL Server on Windows, see [Section 2.3.14, “Upgrading MySQL on Windows”](#).

- If you are using replication, see [Section 6.6, “Upgrading a Replication Setup”](#), for information on upgrading your replication setup.
- If you previously installed a MySQL-Max distribution that includes a server named **mysqld-max**, and then upgrade later to a non-Max version of MySQL, **mysqld_safe** still attempts to run the old **mysqld-max** server. If you perform such an upgrade, you should remove the old **mysqld-max** server manually to ensure that **mysqld_safe** runs the new **mysqld** server.

You can always move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL. If you change the character set when running MySQL, you must run **mysamchk -r -q --set-collation=collation_name** on all MyISAM tables. Otherwise, your indexes may not be ordered correctly, because changing the character set may also change the sort order.

If you are cautious about using new versions, you can always rename your old **mysqld** before installing a newer one. For example, if you are using MySQL 4.1.13 and want to upgrade to 5.0.10, rename your current server from **mysqld** to **mysqld-4.1.13**. If your new **mysqld** then does something unexpected, you can simply shut it down and restart with your old **mysqld**.

If, after an upgrade, you experience problems with recompiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, you should check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries.

If problems occur, such as that the new **mysqld** server does not start or that you cannot connect without a password, verify that you do not have an old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` option (for example, **mysqld --print-defaults**). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.

It is a good idea to rebuild and reinstall the Perl `DBD: :mysql` module whenever you install a new release of MySQL. The same applies to other MySQL interfaces as well, such as the PHP `mysql` extension and the Python `MySQLdb`

module.

2.11.1. Upgrading from MySQL 5.0 to 5.1

When upgrading a 5.0 installation to 5.0.10 or above note that it is *necessary* to upgrade your grant tables. Otherwise, creating stored procedures and functions might not work. The procedure for doing this is described in [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

2.11.2. Upgrading from MySQL 4.1 to 5.0

Note: It is good practice to back up your data before installing any new version of software. Although MySQL works very hard to ensure a high level of quality, you should protect your data by making a backup. MySQL generally recommends that you dump and reload your tables from any previous version to upgrade to 5.0.

In general, you should do the following when upgrading from MySQL 4.1 from 5.0:

- Check the items in the change lists found later in this section to see whether any of them might affect your applications. Note particularly any that are marked **Incompatible change**. These result in incompatibilities with earlier versions of MySQL, and may require your attention *before you upgrade*.
- Some releases of MySQL introduce incompatible changes to tables. (Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.) Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.

To avoid problems due to such changes, after you upgrade to a new version of MySQL, you should check your tables (and repair them if necessary), and update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

- Read the MySQL 5.0 change history to see what significant new features you can use in 5.0. See [Section D.1, “Changes in release 5.0.x](#)

[\(Production\)](#)”.

- If you are running MySQL Server on Windows, see [Section 2.3.14, “Upgrading MySQL on Windows”](#).
- MySQL 5.0 adds support for stored procedures. This support requires the `mysql.proc` table. To create this table, you should run the **mysql_upgrade** program as described in [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).
- MySQL 5.0 adds support for views. This support requires extra privilege columns in the `mysql.user` and `mysql.db` tables. To create these columns, you should run the **mysql_upgrade** program as described in [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).
- If you are using replication, see [Section 6.6, “Upgrading a Replication Setup”](#), for information on upgrading your replication setup.

Several visible behaviors have changed between MySQL 4.1 and MySQL 5.0 to make MySQL more compatible with standard SQL. These changes may affect your applications.

The following lists describe changes that may affect applications and that you should watch out for when upgrading to MySQL 5.0.

Server Changes:

- **Incompatible change:** The indexing order for end-space in TEXT columns for InnoDB and MyISAM tables has changed. Starting from 5.0.3, TEXT indexes are compared as space-padded at the end (just as MySQL sorts CHAR, VARCHAR and TEXT fields). If you have an index on a TEXT column, you should run `CHECK TABLE` on it. If the check reports errors, rebuild the indexes: Dump and reload the table if it is an InnoDB table, or run `OPTIMIZE TABLE` or `REPAIR TABLE` if it is a MyISAM table.
- **Warning: Incompatible change.** For BINARY columns, the pad value and how it is handled has changed as of MySQL 5.0.15. The pad value for inserts now is `0x00` rather than space, and there is no stripping of the pad value for retrievals. For details, see [Section 11.4.2, “The BINARY and VARBINARY Types”](#).

- **Incompatible change:** As of MySQL 5.0.3, the server by default no longer loads user-defined functions (UDFs) unless they have at least one auxiliary symbol (for example, an `xxx_init` or `xxx_deinit` symbol) defined in addition to the main function symbol. This behavior can be overridden with the `--allow-suspicious-udfs` option. See [Section 24.2.4.6, “User-Defined Function Security Precautions”](#).

- **Incompatible change:** The update log has been removed in MySQL 5.0. If you had enabled it previously, you should enable the binary log instead.

- **Incompatible change:** Support for the ISAM storage engine has been removed in MySQL 5.0. If you have any ISAM tables, you should convert them *before* upgrading. For example, to convert an ISAM table to use the MyISAM storage engine, use this statement:

```
ALTER TABLE tbl_name ENGINE = MyISAM;
```

Use a similar statement for every ISAM table in each of your databases.

- **Incompatible change:** Support for RAID options in MyISAM tables has been removed in MySQL 5.0. If you have tables that use these options, you should convert them before upgrading. One way to do this is to dump them with **mysqldump**, edit the dump file to remove the RAID options in the `CREATE TABLE` statements, and reload the dump file. Another possibility is to use `CREATE TABLE new_tbl ... SELECT raid_tbl` to create a new table from the RAID table. However, the `CREATE TABLE` part of the statement must contain sufficient information to re-create column attributes as well as indexes, or column attributes may be lost and indexes will not appear in the new table. See [Section 13.1.5, “CREATE TABLE Syntax”](#).

The `.MYD` files for RAID tables in a given database are stored under the database directory in subdirectories that have names consisting of two hex digits in the range from `00` to `ff`. After converting all tables that use RAID options, these RAID-related subdirectories still will exist but can be removed. Verify that they are empty, and then remove them manually. (If they are not empty, there is some RAID table that has not been converted.)

- In MySQL 5.0.6, binary logging of stored routines and triggers was changed. This change has implications for security, replication, and data recovery, as discussed in [Section 17.4, “Binary Logging of Stored Routines](#)

[and Triggers](#)".

SQL Changes:

- **Incompatible change:** Beginning with MySQL 5.0.12, natural joins and joins with `USING`, including outer join variants, are processed according to the SQL:2003 standard. The changes include elimination of redundant output columns for `NATURAL` joins and joins specified with a `USING` clause and proper ordering of output columns. The precedence of the comma operator also now is lower compared to `JOIN`, `LEFT JOIN`, and so forth.

These changes make MySQL more compliant with standard SQL. However, they can result in different output columns for some joins. Also, some queries that appeared to work correctly prior to 5.0.12 must be rewritten to comply with the standard. For details about the scope of the changes and examples that show what query rewrites are necessary, see [Section 13.2.7.1, "JOIN Syntax"](#).

- **Incompatible change:** Previously, a lock wait timeout caused InnoDB to roll back the entire current transaction. As of MySQL 5.0.13, it rolls back only the most recent SQL statement.
- **Incompatible change:** The namespace for triggers has changed in MySQL 5.0.10. Previously, trigger names had to be unique per table. Now they must be unique within the schema (database). An implication of this change is that `DROP TRIGGER` syntax now uses a schema name instead of a table name (schema name is optional and, if omitted, the current schema will be used).

When upgrading from a previous version of MySQL 5 to MySQL 5.0.10 or newer, you must drop all triggers and re-create them or `DROP TRIGGER` will not work after the upgrade. Here is a suggested procedure for doing this:

1. Upgrade to MySQL 5.0.10 or later to be able to access trigger information in the `INFORMATION_SCHEMA.TRIGGERS` table. (It should work even for pre-5.0.10 triggers.)
2. Dump all trigger definitions using the following `SELECT` statement:

```
SELECT CONCAT('CREATE TRIGGER ', t.TRIGGER_SCHEMA, '.', t.TRIGGER_NAME, ' ', t.ACTION_TIMING, ' ', t.EVENT_MANIPULATION,
```

```

        t.EVENT_OBJECT_SCHEMA, '.', t.EVENT_OBJECT_TABLE,
        ' FOR EACH ROW ', t.ACTION_STATEMENT, '///' )
INTO OUTFILE '/tmp/triggers.sql'
FROM INFORMATION_SCHEMA.TRIGGERS AS t;

```

The statement uses INTO OUTFILE, so you must have the FILE privilege. The file will be created on the server host. Use a different filename if you like. To be 100% safe, inspect the trigger definitions in the triggers.sql file, and perhaps make a backup of the file.

3. Stop the server and drop all triggers by removing all .TRG files in your database directories. Change location to your data directory and issue this command:

```
shell> rm */*.TRG
```

4. Start the server and re-create all triggers using the triggers.sql file. For the file created earlier, use these commands in the **mysql** program:

```
mysql> delimiter // ;
mysql> source /tmp/triggers.sql //
```

5. Use the SHOW TRIGGERS statement to check that all triggers were created successfully.

- **Incompatible change:** As of MySQL 5.0.15, the CHAR() function returns a binary string rather than a string in the connection character set. An optional USING charset_name clause may be used to produce a result in a specific character set instead. Also, arguments larger than 256 produce multiple characters. They are no longer interpreted modulo 256 to produce a single character each. These changes may cause some incompatibilities:

- CHAR(ORD('A')) = 'a' is no longer true:

```
mysql> SELECT CHAR(ORD('A')) = 'a';
+-----+
| CHAR(ORD('A')) = 'a' |
+-----+
|                        0 |
+-----+
```

To perform a case-insensitive comparison, you can produce a result string in a non-binary character set by adding a USING clause or

converting the result:

```
mysql> SELECT CHAR(ORD('A') USING latin1) = 'a';
+-----+
| CHAR(ORD('A') USING latin1) = 'a' |
+-----+
|                                     1 |
+-----+
mysql> SELECT CONVERT(CHAR(ORD('A')) USING latin1) = 'a';
+-----+
| CONVERT(CHAR(ORD('A')) USING latin1) = 'a' |
+-----+
|                                     1 |
+-----+
```

- CREATE TABLE ... SELECT CHAR(...) produces a VARBINARY column, not a VARCHAR column. To produce a VARCHAR column, use USING or CONVERT() as just described to convert the CHAR() result into a non-binary character set.
- Previously, the following statements inserted the value 0x00410041 ('AA' as a ucs2 string) into the table:

```
CREATE TABLE t (ucs2_column CHAR(2) CHARACTER SET ucs2);
INSERT INTO t VALUES (CHAR(0x41,0x41));
```

As of MySQL 5.0.15, the statements insert a single ucs2 character with value 0x4141.

- **Incompatible change:** By default, integer subtraction involving an unsigned value should produce an unsigned result. Tracking of the “unsignedness” of an expression was improved in MySQL 5.0.13. This means that, in some cases where an unsigned subtraction would have resulted in a signed integer, it now results in an unsigned integer. One context in which this difference manifests itself is when a subtraction involving an unsigned operand would be negative.

Suppose that *i* is a TINYINT UNSIGNED column and has a value of 0. The server evaluates the following expression using 64-bit unsigned integer arithmetic with the following result:

```
mysql> SELECT i - 1 FROM t;
+-----+
```

```

| i - 1 |
+-----+
| 18446744073709551615 |
+-----+

```

If the expression is used in an `UPDATE t SET i = i - 1` statement, the expression is evaluated and the result assigned to `i` according to the usual rules for handling values outside the column range or 0 to 255. That is, the value is clipped to the nearest endpoint of the range. However, the result is version-specific:

- Before MySQL 5.0.13, the expression is evaluated but is treated as the equivalent 64-bit signed value (-1) for the assignment. The value of -1 is clipped to the nearest endpoint of the column range, resulting in a value of 0:

```

mysql> UPDATE t SET i = i - 1; SELECT i FROM t;
+-----+
| i     |
+-----+
|      0 |
+-----+

```

- As of MySQL 5.0.13, the expression is evaluated and retains its unsigned attribute for the assignment. The value of 18446744073709551615 is clipped to the nearest endpoint of the column range, resulting in a value of 255:

```

mysql> UPDATE t SET i = i - 1; SELECT i FROM t;
+-----+
| i     |
+-----+
|    255 |
+-----+

```

To get the older behavior, use `CAST()` to convert the expression result to a signed value:

```
UPDATE t SET i = CAST(i - 1 AS SIGNED);
```

Alternatively, set the `NO_UNSIGNED_SUBTRACTION` SQL mode. However, this will affect all integer subtractions involving unsigned values.

- **Incompatible change:** Before MySQL 5.0.13, `NOW()` and `SYSDATE()` return

the same value (the time at which the statement in which the function occurs begins executing). As of MySQL 5.0.13, `SYSDATE()` returns the time at which it executes, which can differ from the value returned by `NOW()`. For information about the implications for binary logging and replication, see the description for `SYSDATE()` in [Section 12.5, “Date and Time Functions”](#) and for `SET TIMESTAMP` in [Section 13.5.3, “SET Syntax”](#). To restore the former behavior for `SYSDATE()` and cause it to be an alias for `NOW()`, start the server with the `--sysdate-is-now` option (available as of MySQL 5.0.20).

- **Incompatible change:** Before MySQL 5.0.13, `GREATEST(x,NULL)` and `LEAST(x,NULL)` return `x` when `x` is a non-NULL value. As of 5.0.3, both functions return NULL if any argument is NULL, the same as Oracle. This change can cause problems for applications that rely on the old behavior.
- **Incompatible change:** Before MySQL 4.1.13/5.0.8, conversion of DATETIME values to numeric form by adding zero produced a result in `YYYYMMDDHHMMSS` format. The result of `DATETIME+0` is now in `YYYYMMDDHHMMSS.000000` format.
- **Incompatible change:** In MySQL 4.1.12/5.0.6, the behavior of `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` has changed when the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values both are empty. Formerly, a column was read or written the display width of the column. For example, `INT(4)` was read or written using a field with a width of 4. Now columns are read and written using a field width wide enough to hold all values in the field. However, data files written before this change was made might not be reloaded correctly with `LOAD DATA INFILE` for MySQL 4.1.12/5.0.6 and up. This change also affects data files read by **mysqlimport** and written by **mysqldump --tab**, which use `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`. For more information, see [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).
- **Incompatible change:** The implementation of DECIMAL has changed in MySQL 5.0.3. You should make your applications aware of this change. For information about this change, and about possible incompatibilities with old applications, see [Chapter 21, Precision Math](#).

DECIMAL columns are stored in a more efficient format. To convert a table to

use the new `DECIMAL` type, you should do an `ALTER TABLE` on it. (The `ALTER TABLE` also will change the table's `VARCHAR` columns to use the new `VARCHAR` data type properties, described in a separate item.)

A consequence of the change in handling of the `DECIMAL` and `NUMERIC` fixed-point data types is that the server is more strict to follow standard SQL. For example, a data type of `DECIMAL(3,1)` stores a maximum value of 99.9. Before MySQL 5.0.3, the server allowed larger numbers to be stored. That is, it stored a value such as 100.0 as 100.0. As of MySQL 5.0.3, the server clips 100.0 to the maximum allowable value of 99.9. If you have tables that were created before MySQL 5.0.3 and that contain floating-point data not strictly legal for the data type, you should alter the data types of those columns. For example:

```
ALTER TABLE tbl_name MODIFY col_name DECIMAL(4,1);
```

The behavior used by the server for `DECIMAL` columns in a table depends on the version of MySQL used to create the table. If your server is from MySQL 5.0.3 or higher, but you have `DECIMAL` columns in tables that were created before 5.0.3, the old behavior still applies to those columns. To convert the tables to the newer `DECIMAL` format, dump them with **mysqldump** and reload them.

- **Incompatible change:** MySQL 5.0.3 and up uses precision math when calculating with `DECIMAL` and integer columns (64 decimal digits) and for rounding exact-value numbers. Rounding behavior is well-defined, not dependent on the implementation of the underlying C library. However, this might result in incompatibilities for applications that rely on the old behavior. (For example, inserting .5 into an `INT` column results in 1 as of MySQL 5.0.3, but might be 0 in older versions.) For more information about rounding behavior, see [Section 21.4, “Rounding Behavior”](#), and [Section 21.5, “Precision Math Examples”](#).
- **Incompatible change:** MyISAM and InnoDB tables created with `DECIMAL` columns in MySQL 5.0.3 to 5.0.5 will appear corrupt after an upgrade to MySQL 5.0.6. (The same incompatibility will occur for these tables created in MySQL 5.0.6 after a downgrade to MySQL 5.0.3 to 5.0.5.) If you have such tables, check and repair them with **mysql_upgrade** after upgrading. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

- **Incompatible change:** Before MySQL 5.0.2, `SHOW STATUS` returned global status values. The default as of 5.0.2 is to return session values, which is incompatible with previous versions. To issue a `SHOW STATUS` statement that will retrieve global status values for all versions of MySQL, write it like this:

```
SHOW /*!50002 GLOBAL */ STATUS;
```

- **Incompatible change:** User variables are not case sensitive in MySQL 5.0. In MySQL 4.1, `SET @x = 0; SET @X = 1; SELECT @x;` created two variables and returned 0. In MySQL 5.0, it creates one variable and returns 1. Replication setups that rely on the old behavior may be affected by this change.
- Some keywords are reserved in MySQL 5.0 that were not reserved in MySQL 4.1. See [Section 9.5, “Treatment of Reserved Words in MySQL”](#).
- As of MySQL 5.0.3, trailing spaces no longer are removed from values stored in `VARCHAR` and `VARBINARY` columns. The maximum lengths for `VARCHAR` and `VARBINARY` columns in MySQL 5.0.3 and later are 65,535 characters and 65,535 bytes, respectively.

When a binary upgrade (filesystem-level copy of data files) to MySQL 5.0 is performed for a table with a `VARBINARY` column, the column is space-padded to the full allowable width of the column. This causes values in `VARBINARY` columns that do not occupy the full width of the column to include extra trailing spaces after the upgrade, which means that the data in the column is different.

In addition, new rows inserted into a table upgraded in this way will be space padded to the full width of the column.

This issue can be resolved as follows:

1. For each table containing `VARBINARY` columns, execute the statement

```
ALTER TABLE table_name ENGINE=engine_name;
```

where *table_name* is the name of the table and *engine_name* is the name of the storage engine currently used by *table_name*. In other

words, if the table named `mytable` uses the MyISAM storage engine, then you would use this statement:

```
ALTER TABLE mytable ENGINE=MYISAM;
```

This rebuilds the table so that it uses the 5.0 VARBINARY format.

2. Then you must remove all trailing spaces from any VARBINARY column values. For each VARBINARY column *varbinary_column*, you should perform the following statement (where *table_name* is the name of the table containing the VARBINARY column):

```
UPDATE table_name SET varbinary_column = RTRIM(varbinary_co:
```

This is necessary and safe because trailing spaces are stripped before 5.0.3, meaning that any trailing spaces are erroneous.

This problem does not occur (and thus these two steps are not required) for tables upgraded using the recommended procedure of dumping tables prior to the upgrade and reloading them afterwards.

Note: If you create a table with new VARCHAR or VARBINARY columns in MySQL 5.0.3 or later, the table will not be usable if you downgrade to a version older than 5.0.3. Dump the table with **mysqldump** before downgrading and reload it after downgrading.

- Comparisons made between FLOAT or DOUBLE values that happened to work in MySQL 4.1 may not do so in 5.0. Values of these types are imprecise in all MySQL versions, and you are *strongly advised* to avoid such comparisons as `WHERE col_name=some_double`, regardless of the MySQL version you are using. See [Section A.5.8, “Problems with Floating-Point Comparisons”](#).
- As of MySQL 5.0.3, BIT is a separate data type, not a synonym for TINYINT(1). See [Section 11.1.1, “Overview of Numeric Types”](#).
- MySQL 5.0.2 adds several SQL modes that allow stricter control over rejecting records that have invalid or missing values. See [Section 5.2.5, “The Server SQL Mode”](#), and [Section 1.9.6.2, “Constraints on Invalid Data”](#). If you want to enable this control but continue to use MySQL's

capability for storing incorrect dates such as '2004-02-31', you should start the server with `--sql_mode="TRADITIONAL,ALLOW_INVALID_DATES"`.

- As of MySQL 5.0.2, the `SCHEMA` and `SCHEMAS` keywords are accepted as synonyms for `DATABASE` and `DATABASES`, respectively. (While “schemata” is grammatically correct and even appears in some MySQL 5.0 system database and table names, it cannot be used as a keyword.)
- A new startup option named `innodb_table_locks` was added that causes `LOCK TABLE` to also acquire InnoDB table locks. This option is enabled by default. This can cause deadlocks in applications that use `AUTOCOMMIT=1` and `LOCK TABLES`. If your application encounters deadlocks after upgrading, you may need to add `innodb_table_locks=0` to your `my.cnf` file.

C API Changes:

- **Incompatible change:** Because the MySQL 5.0 server has a new implementation of the `DECIMAL` data type, a problem may occur if the server is used by older clients that still are linked against MySQL 4.1 client libraries. If a client uses the binary client/server protocol to execute prepared statements that generate result sets containing numeric values, an error will be raised: 'Using unsupported buffer type: 246'

This error occurs because the 4.1 client libraries do not support the new `MYSQL_TYPE_NEWDECIMAL` type value added in 5.0. There is no way to disable the new `DECIMAL` data type on the server side. You can avoid the problem by relinking the application with the client libraries from MySQL 5.0.

- **Incompatible change:** The `ER_WARN_DATA_TRUNCATED` warning symbol was renamed to `WARN_DATA_TRUNCATED` in MySQL 5.0.3.
- The `reconnect` flag in the `MYSQL` structure is set to 0 by `mysql_real_connect()`. Only those client programs which did not explicitly set this flag to 0 or 1 after `mysql_real_connect()` experience a change. Having automatic reconnection enabled by default was considered too dangerous (due to the fact that table locks, temporary tables, user variables, and session variables are lost after reconnection).

2.11.3. Copying MySQL Databases to Another Machine

You can copy the `.frm`, `.MYI`, and `.MYD` files for MyISAM tables between different architectures that support the same floating-point format. (MySQL takes care of any byte-swapping issues.) See [Section 14.1, “The MyISAM Storage Engine”](#).

In cases where you need to transfer databases between different architectures, you can use **mysqldump** to create a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the **mysql** client.

Use **mysqldump --help** to see what options are available. If you are moving the data to a newer version of MySQL, you should use **mysqldump --opt** to take advantage of any optimizations that result in a dump file that is smaller and can be processed more quickly.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump --opt db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --opt --compress db_name | mysq
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
shell> mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
shell> mysqladmin create db_name
shell> gunzip < db_name.gz | mysql db_name
```

You can also use **mysqldump** and **mysqlimport** to transfer the database. For

large tables, this is much faster than simply using **mysqldump**. In the following commands, *DUMPDIR* represents the full pathname of the directory you use to store the output from **mysqldump**.

First, create the directory for the output files and dump the database:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the *DUMPDIR* directory to some corresponding directory on the target machine and load the files into MySQL there:

```
shell> mysqladmin create db_name # create database
shell> cat DUMPDIR/*.sql | mysql db_name # create tables in databa
shell> mysqlimport db_name DUMPDIR/*.txt # load data into tables
```

Do not forget to copy the `mysql` database because that is where the grant tables are stored. You might have to run commands as the MySQL root user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute **mysqladmin flush-privileges** so that the server reloads the grant table information.

2.12. Downgrading MySQL

This section describes what you should do to downgrade to an older MySQL version in the unlikely case that the previous version worked better than the new one.

If you are downgrading within the same release series (for example, from 4.1.13 to 4.1.12) the general rule is that you just have to install the new binaries on top of the old ones. There is no need to do anything with the databases. As always, however, it is always a good idea to make a backup.

The following items form a checklist of things you should do whenever you perform a downgrade:

- Read the upgrading section for the release series from which you are downgrading to be sure that it does not have any features you really need. [Section 2.11, “Upgrading MySQL”](#).
- If there is a downgrading section for that version, you should read that as well.

In most cases, you can move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL.

If you downgrade from one release series to another, there may be incompatibilities in table storage formats. In this case, you can use **mysqldump** to dump your tables before downgrading. After downgrading, reload the dump file using **mysql** or **mysqlimport** to re-create your tables. For examples, see [Section 2.11.3, “Copying MySQL Databases to Another Machine”](#).

The normal symptom of a downward-incompatible table format change when you downgrade is that you can't open tables. In that case, use the following procedure:

1. Stop the older MySQL server that you are downgrading to.
2. Restart the newer MySQL server you are downgrading from.

3. Dump any tables that were inaccessible to the older server by using **mysqldump** to create a dump file.
4. Stop the newer MySQL server and restart the older one.
5. Reload the dump file into the older server. Your tables should be accessible.

2.12.1. Downgrading to MySQL 4.1

MySQL 4.1 does not support stored routines or triggers. If your databases contain stored routines or triggers, prevent them from being dumped when you use **mysqldump** by using the `--skip-routines` and `--skip-triggers` options. (See [Section 8.12, “mysqldump — A Database Backup Program”](#).)

MySQL 4.1 does not support views. If your databases contain views, remove them with `DROP VIEW` before using **mysqldump**. (See [Section 19.3, “DROP VIEW Syntax”](#).)

After downgrading from MySQL 5.0, you may see the following information in the `mysql.err` file:

```
Incorrect information in file: './mysql/user.frm'
```

In this case, you can do the following:

1. Start MySQL 5.0.4 (or newer).
2. Run **mysql_fix_privilege_tables**, which will change the `mysql.user` table to a format that both MySQL 4.1 and 5.0 can use.
3. Stop the MySQL server.
4. Start MySQL 4.1.

If the preceding procedure fails, you should be able to do the following instead:

1. Start MySQL 5.0.4 (or newer).
2. Run **mysqldump --opt --add-drop-table mysql > /tmp/mysql.dump**.
3. Stop the MySQL server.

4. Start MySQL 4.1 with the `--skip-grant` option.
5. Run **`mysql mysql < /tmp/mysql.dump`**.
6. Run **`mysqladmin flush-privileges`**.

2.13. Operating System-Specific Notes

2.13.1. Linux Notes

This section discusses issues that have been found to occur on Linux. The first few subsections describe general operating system-related issues, problems that can occur when using binary or source distributions, and post-installation issues. The remaining subsections discuss problems that occur with Linux on specific platforms.

Note that most of these problems occur on older versions of Linux. If you are running a recent version, you may see none of them.

2.13.1.1. Linux Operating System Notes

MySQL needs at least Linux version 2.0.

Warning: We have seen some strange problems with Linux 2.2.14 and MySQL on SMP systems. We also have reports from some MySQL users that they have encountered serious stability problems using MySQL with kernel 2.2.14. If you are using this kernel, you should upgrade to 2.2.19 (or newer) or to a 2.4 kernel. If you have a multiple-CPU box, you should seriously consider using 2.4 because it gives you a significant speed boost. Your system should be more stable.

When using LinuxThreads, you should see a minimum of three **mysqld** processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

2.13.1.2. Linux Binary Distribution Notes

The Linux-Intel binary and RPM releases of MySQL are configured for the highest possible speed. We are always trying to use the fastest stable compiler available.

The binary release is linked with `-static`, which means you do not normally

need to worry about which version of the system libraries you have. You need not install LinuxThreads, either. A program linked with `-static` is slightly larger than a dynamically linked program, but also slightly faster (3-5%). However, one problem with a statically linked program is that you can't use user-defined functions (UDFs). If you are going to write or use UDFs (this is something for C or C++ programmers only), you must compile MySQL yourself using dynamic linking.

A known issue with binary distributions is that on older Linux systems that use `libc` (such as Red Hat 4.x or Slackware), you get some (non-fatal) issues with hostname resolution. If your system uses `libc` rather than `glibc2`, you probably will encounter some difficulties with hostname resolution and `getpwnam()`. This happens because `glibc` (unfortunately) depends on some external libraries to implement hostname resolution and `getpwent()`, even when compiled with `-static`. These problems manifest themselves in two ways:

- You may see the following error message when you run **mysql_install_db**:

```
Sorry, the host 'xxxx' could not be looked up
```

You can deal with this by executing **mysql_install_db --force**, which does not execute the **resolveip** test in **mysql_install_db**. The downside is that you cannot use hostnames in the grant tables: except for `localhost`, you must use IP numbers instead. If you are using an old version of MySQL that does not support `--force`, you must manually remove the **resolveip** test in `mysql_install` using a text editor.

- You also may see the following error when you try to run **mysqld** with the `--user` option:

```
getpwnam: No such file or directory
```

To work around this problem, start **mysqld** by using the `su` command rather than by specifying the `--user` option. This causes the system itself to change the user ID of the **mysqld** process so that **mysqld** need not do so.

Another solution, which solves both problems, is not to use a binary distribution. Obtain a MySQL source distribution (in RPM or `tar.gz` format) and install that instead.

On some Linux 2.2 versions, you may get the error `Resource temporarily unavailable` when clients make a great many new connections to a **mysqld** server over TCP/IP. The problem is that Linux has a delay between the time that you close a TCP/IP socket and the time that the system actually frees it. There is room for only a finite number of TCP/IP slots, so you encounter the resource-unavailable error if clients attempt too many new TCP/IP connections over a short period of time. For example, you may see the error when you run the MySQL `test-connect` benchmark over TCP/IP.

We have inquired about this problem a few times on different Linux mailing lists but have never been able to find a suitable resolution. The only known “fix” is for clients to use persistent connections, or, if you are running the database server and clients on the same machine, to use Unix socket file connections rather than TCP/IP connections.

2.13.1.3. Linux Source Distribution Notes

The following notes regarding `glibc` apply only to the situation when you build MySQL yourself. If you are running Linux on an x86 machine, in most cases it is much better for you to use our binary. We link our binaries against the best patched version of `glibc` we can find and with the best compiler options, in an attempt to make it suitable for a high-load server. For a typical user, even for setups with a lot of concurrent connections or tables exceeding the 2GB limit, our binary is the best choice in most cases. After reading the following text, if you are in doubt about what to do, try our binary first to determine whether it meets your needs. If you discover that it is not good enough, you may want to try your own build. In that case, we would appreciate a note about it so that we can build a better binary next time.

MySQL uses `LinuxThreads` on Linux. If you are using an old Linux version that doesn't have `glibc2`, you must install `LinuxThreads` before trying to compile MySQL. You can obtain `LinuxThreads` from <http://dev.mysql.com/downloads/os-linux.html>.

Note that `glibc` versions before and including version 2.1.1 have a fatal bug in `pthread_mutex_timedwait()` handling, which is used when `INSERT DELAYED` statements are issued. We recommend that you not use `INSERT DELAYED` before upgrading `glibc`.

Note that Linux kernel and the LinuxThread library can by default handle a maximum of 1,024 threads. If you plan to have more than 1,000 concurrent connections, you need to make some changes to LinuxThreads, as follows:

- Increase `PTHREAD_THREADS_MAX` in `sysdeps/unix/sysv/linux/bits/local_lim.h` to 4096 and decrease `STACK_SIZE` in `linuxthreads/internals.h` to 256KB. The paths are relative to the root of `glibc`. (Note that MySQL is not stable with 600-1000 connections if `STACK_SIZE` is the default of 2MB.)
- Recompile LinuxThreads to produce a new `libpthread.a` library, and relink MySQL against it.

Additional information about circumventing thread limits in LinuxThreads can be found at <http://www.volano.com/linuxnotes.html>.

There is another issue that greatly hurts MySQL performance, especially on SMP systems. The mutex implementation in LinuxThreads in `glibc 2.1` is very poor for programs with many threads that hold the mutex only for a short time. This produces a paradoxical result: If you link MySQL against an unmodified LinuxThreads, removing processors from an SMP actually improves MySQL performance in many cases. We have made a patch available for `glibc 2.1.3` to correct this behavior (<http://www.mysql.com/Downloads/Linux/linuxthreads-2.1-patch>).

With `glibc 2.2.2`, MySQL uses the adaptive mutex, which is much better than even the patched one in `glibc 2.1.3`. Be warned, however, that under some conditions, the current mutex code in `glibc 2.2.2` overspins, which hurts MySQL performance. The likelihood that this condition occurs can be reduced by re-nicing the **mysqld** process to the highest priority. We have also been able to correct the overspin behavior with a patch, available at <http://www.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch>. It combines the correction of overspin, maximum number of threads, and stack spacing all in one. You need to apply it in the `linuxthreads` directory with `patch -p0 </tmp/linuxthreads-2.2.2.patch`. We hope it is included in some form in future releases of `glibc 2.2`. In any case, if you link against `glibc 2.2.2`, you still need to correct `STACK_SIZE` and `PTHREAD_THREADS_MAX`. We hope that the defaults is corrected to some more acceptable values for high-load MySQL setup in the future, so that the commands needed to produce your own build can be

reduced to **./configure; make; make install**.

We recommend that you use these patches to build a special static version of `libpthread.a` and use it only for statically linking against MySQL. We know that these patches are safe for MySQL and significantly improve its performance, but we cannot say anything about their effects on other applications. If you link other applications that require LinuxThreads against the patched static version of the library, or build a patched shared version and install it on your system, you do so at your own risk.

If you experience any strange problems during the installation of MySQL, or with some common utilities hanging, it is very likely that they are either library or compiler related. If this is the case, using our binary resolves them.

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqlclient.so` to `/usr/lib`.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you are using the Fujitsu compiler (`fcc/FCC`), you may have some problems compiling MySQL because the Linux header files are very `gcc` oriented. The following **configure** line should work with **fcc/FCC**:

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE \
-DCONST=const -DNO_STRTOLL_PROTO" \
CXX=FCC CXXFLAGS="-O -K fast -K lib \
-K omitfp -K preex --no_exceptions --no_rtti -D_GNU_SOURCE \
-DCONST=const -Dalloca=__builtin_alloca -DNO_STRTOLL_PROTO \
'-D_EXTERN_INLINE=static __inline'" \
./configure \
--prefix=/usr/local/mysql --enable-asm \
```

```
--with-mysqld-ldflags=-all-static --disable-shared \  
--with-low-memory
```

2.13.1.4. Linux Post-Installation Notes

mysql.server can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).

If MySQL cannot open enough files or connections, it may be that you have not configured Linux to handle enough files.

In Linux 2.2 and onward, you can check the number of allocated file handles as follows:

```
shell> cat /proc/sys/fs/file-max  
shell> cat /proc/sys/fs/dquot-max  
shell> cat /proc/sys/fs/super-max
```

If you have more than 16MB of memory, you should add something like the following to your init scripts (for example, `/etc/init.d/boot.local` on SuSE Linux):

```
echo 65536 > /proc/sys/fs/file-max  
echo 8192 > /proc/sys/fs/dquot-max  
echo 1024 > /proc/sys/fs/super-max
```

You can also run the echo commands from the command line as root, but these settings are lost the next time your computer restarts.

Alternatively, you can set these parameters on startup by using the `sysctl` tool, which is used by many Linux distributions (including SuSE Linux 8.0 and later). Put the following values into a file named `/etc/sysctl.conf`:

```
# Increase some values for MySQL  
fs.file-max = 65536  
fs.dquot-max = 8192  
fs.super-max = 1024
```

You should also add the following to `/etc/my.cnf`:

```
[mysqld_safe]  
open-files-limit=8192
```

This should allow the server a limit of 8,192 for the combined number of connections and open files.

The `STACK_SIZE` constant in LinuxThreads controls the spacing of thread stacks in the address space. It needs to be large enough so that there is plenty of room for each individual thread stack, but small enough to keep the stack of some threads from running into the global **mysqld** data. Unfortunately, as we have experimentally discovered, the Linux implementation of `mmap()` successfully unmaps a mapped region if you ask it to map out an address currently in use, zeroing out the data on the entire page instead of returning an error. So, the safety of **mysqld** or any other threaded application depends on the “gentlemanly” behavior of the code that creates threads. The user must take measures to make sure that the number of running threads at any given time is sufficiently low for thread stacks to stay away from the global heap. With **mysqld**, you should enforce this behavior by setting a reasonable value for the `max_connections` variable.

If you build MySQL yourself, you can patch LinuxThreads for better stack use. See [Section 2.13.1.3, “Linux Source Distribution Notes”](#). If you do not want to patch LinuxThreads, you should set `max_connections` to a value no higher than 500. It should be even less if you have a large key buffer, large heap tables, or some other things that make **mysqld** allocate a lot of memory, or if you are running a 2.2 kernel with a 2GB patch. If you are using our binary or RPM version, you can safely set `max_connections` at 1500, assuming no large key buffer or heap tables with lots of data. The more you reduce `STACK_SIZE` in LinuxThreads the more threads you can safely create. We recommend values between 128KB and 256KB.

If you use a lot of concurrent connections, you may suffer from a “feature” in the 2.2 kernel that attempts to prevent fork bomb attacks by penalizing a process for forking or cloning a child. This causes MySQL not to scale well as you increase the number of concurrent clients. On single-CPU systems, we have seen this manifest as very slow thread creation; it may take a long time to connect to MySQL (as long as one minute), and it may take just as long to shut it down. On multiple-CPU systems, we have observed a gradual drop in query speed as the number of clients increases. In the process of trying to find a solution, we have received a kernel patch from one of our users who claimed it helped for his site.

This patch is available at <http://www.mysql.com/Downloads/Patches/linux-fork.patch>. We have done rather extensive testing of this patch on both development and production systems. It has significantly improved MySQL performance without causing any problems and we recommend it to our users who still run high-load servers on 2.2 kernels.

This issue has been fixed in the 2.4 kernel, so if you are not satisfied with the current performance of your system, rather than patching your 2.2 kernel, it might be easier to upgrade to 2.4. On SMP systems, upgrading also gives you a nice SMP boost in addition to fixing the fairness bug.

We have tested MySQL on the 2.4 kernel on a two-CPU machine and found MySQL scales *much* better. There was virtually no slowdown on query throughput all the way up to 1,000 clients, and the MySQL scaling factor (computed as the ratio of maximum throughput to the throughput for one client) was 180%. We have observed similar results on a four-CPU system: Virtually no slowdown as the number of clients was increased up to 1,000, and a 300% scaling factor. Based on these results, for a high-load SMP server using a 2.2 kernel, we definitely recommend upgrading to the 2.4 kernel at this point.

We have discovered that it is essential to run the **mysqld** process with the highest possible priority on the 2.4 kernel to achieve maximum performance. This can be done by adding a `renice -20 $$` command to **mysqld_safe**. In our testing on a four-CPU machine, increasing the priority resulted in a 60% throughput increase with 400 clients.

We are currently also trying to collect more information on how well MySQL performs with a 2.4 kernel on four-way and eight-way systems. If you have access such a system and have done some benchmarks, please send an email message to <benchmarks@mysql.com> with the results. We will review them for inclusion in the manual.

If you see a dead **mysqld** server process with **ps**, this usually means that you have found a bug in MySQL or you have a corrupted table. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#).

To get a core dump on Linux if **mysqld** dies with a SIGSEGV signal, you can start **mysqld** with the `--core-file` option. Note that you also probably need to raise the core file size by adding `ulimit -c 1000000` to **mysqld_safe** or starting

mysqld_safe with `--core-file-size=1000000`. See [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#).

2.13.1.5. Linux x86 Notes

MySQL requires `libc` 5.4.12 or newer. It is known to work with `libc` 5.4.46. `glibc` 2.0.6 and later should also work. There have been some problems with the `glibc` RPMs from Red Hat, so if you have problems, check whether there are any updates. The `glibc` 2.0.7-19 and 2.0.7-29 RPMs are known to work.

If you are using Red Hat 8.0 or a new `glibc` 2.2.x library, you may see **mysqld** die in `gethostbyaddr()`. This happens because the new `glibc` library requires a stack size greater than 128KB for this call. To fix the problem, start **mysqld** with the `--thread-stack=192K` option. (Use `-o thread_stack=192K` before MySQL 4.) This stack size is the default on MySQL 4.0.10 and above, so you should not see the problem.

If you are using `gcc` 3.0 and above to compile MySQL, you must install the `libstdc++v3` library before compiling MySQL; if you don't do this, you get an error about a missing `__cxa_pure_virtual` symbol during linking.

On some older Linux distributions, **configure** may produce an error like this:

```
Syntax error in sched.h. Change _P to __P in the
/usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Just do what the error message says. Add an extra underscore to the `_P` macro name that has only one underscore, and then try again.

You may get some warnings when compiling. Those shown here can be ignored:

```
mysqld.cc -o objs-thread/mysqld.o
mysqld.cc: In function `void init_signals()':
mysqld.cc:315: warning: assignment of negative value `-1' to
`long unsigned int'
mysqld.cc: In function `void * signal_hand(void *)':
mysqld.cc:346: warning: assignment of negative value `-1' to
`long unsigned int'
```

If **mysqld** always dumps core when it starts, the problem may be that you have an old `/lib/libc.a`. Try renaming it, and then remove `sql/mysqld` and do a new

make install and try again. This problem has been reported on some Slackware installations.

If you get the following error when linking **mysqld**, it means that your `libg++.a` is not installed correctly:

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':  
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

You can avoid using `libg++.a` by running **configure** like this:

```
shell> CXX=gcc ./configure
```

2.13.1.6. Linux SPARC Notes

In some implementations, `readdir_r()` is broken. The symptom is that the `SHOW DATABASES` statement always returns an empty set. This can be fixed by removing `HAVE_READDIR_R` from `config.h` after configuring and before compiling.

2.13.1.7. Linux Alpha Notes

We have tested MySQL 5.0 on Alpha with our benchmarks and test suite, and it appears to work well.

We currently build the MySQL binary packages on SuSE Linux 7.0 for AXP, kernel 2.4.4-SMP, Compaq C compiler (V6.2-505) and Compaq C++ compiler (V6.3-006) on a Compaq DS20 machine with an Alpha EV6 processor.

You can find the preceding compilers at <http://www.support.compaq.com/alpha-tools/>. By using these compilers rather than `gcc`, we get about 9-14% better MySQL performance.

For MySQL on Alpha, we use the `-arch generic` flag to our compile options, which ensures that the binary runs on all Alpha processors. We also compile statically to avoid library problems. The **configure** command looks like this:

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx \  
CXXFLAGS="-fast -arch generic -noexceptions -nortti" \  
./configure --prefix=/usr/local/mysql --disable-shared \  
--with-extra-charsets=complex --enable-thread-safe-client \  
--with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_sha
```

If you want to use **egcs**, the following **configure** line worked for us:

```
CFLAGS="-O3 -fomit-frame-pointer" CXX=gcc \  
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \  
-fno-exceptions -fno-rtti" \  
./configure --prefix=/usr/local/mysql --disable-shared
```

Some known problems when running MySQL on Linux-Alpha:

- Debugging threaded applications like MySQL does not work with `gdb` 4.18. You should use **gdb** 5.1 instead.
- If you try linking **mysqld** statically when using **gcc**, the resulting image dumps core at startup time. In other words, *do not* use `--with-mysqld-ldflags=-all-static` with **gcc**.

2.13.1.8. Linux PowerPC Notes

MySQL should work on MkLinux with the newest `glibc` package (tested with `glibc 2.0.7`).

2.13.1.9. Linux MIPS Notes

To get MySQL to work on Qube2 (Linux Mips), you need the newest `glibc` libraries. `glibc-2.0.7-29C2` is known to work. You must also use the **egcs** C++ compiler (**egcs** 1.0.2-9, **gcc** 2.95.2 or newer).

2.13.1.10. Linux IA-64 Notes

To get MySQL to compile on Linux IA-64, we use the following **configure** command for building with **gcc** 2.96:

```
CC=gcc \  
CFLAGS="-O3 -fno-omit-frame-pointer" \  
CXX=gcc \  
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \  
-fno-exceptions -fno-rtti" \  
./configure --prefix=/usr/local/mysql \  
"--with-comment=Official MySQL binary" \  
--with-extra-charsets=complex
```

On IA-64, the MySQL client binaries use shared libraries. This means that if you install our binary distribution at a location other than `/usr/local/mysql`, you need to add the path of the directory where you have `libmysqlclient.so` installed either to the `/etc/ld.so.conf` file or to the value of your `LD_LIBRARY_PATH` environment variable.

See [Section A.3.1, “Problems Linking to the MySQL Client Library”](#).

2.13.1.11. SELinux Notes

RHEL4 comes with SELinux, which supports tighter access control for processes. If SELinux is enabled (`SELINUX` in `/etc/selinux/config` is set to enforcing, `SELINUXTYPE` is set to either `targeted` or `strict`), you might encounter problems installing MySQL AB RPM packages.

Red Hat has an update that solves this. It involves an update of the “security policy” specification to handle the install structure of the RPMs provided by MySQL AB. For further information, see https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=167551 and <http://rhn.redhat.com/errata/RHBA-2006-0049.html>.

2.13.2. Mac OS X Notes

On Mac OS X, `tar` cannot handle long filenames. If you need to unpack a `.tar.gz` distribution, use `gnutar` instead.

2.13.2.1. Mac OS X 10.x (Darwin)

MySQL should work without major problems on Mac OS X 10.x (Darwin).

Known issues:

- If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to `mysqld`. This runs all threads with the same priority. On Mac OS X, this gives better performance, at least until Apple fixes its thread scheduler.
- The connection times (`wait_timeout`, `interactive_timeout` and `net_read_timeout`) values are not honored.

This is probably a signal handling problem in the thread library where the signal doesn't break a pending read and we hope that a future update to the thread libraries will fix this.

Our binary for Mac OS X is compiled on Darwin 6.3 with the following **configure** line:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc \  
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \  
-fno-exceptions -fno-rtti" \  
./configure --prefix=/usr/local/mysql \  
--with-extra-charsets=complex --enable-thread-safe-client \  
--enable-local-infile --disable-shared
```

See [Section 2.5, “Installing MySQL on Mac OS X”](#).

2.13.2.2. Mac OS X Server 1.2 (Rhapsody)

For current versions of Mac OS X Server, no operating system changes are necessary before compiling MySQL. Compiling for the Server platform is the same as for the client version of Mac OS X.

For older versions (Mac OS X Server 1.2, a.k.a. Rhapsody), you must first install a pthread package before trying to configure MySQL.

See [Section 2.5, “Installing MySQL on Mac OS X”](#).

2.13.3. Solaris Notes

For information about installing MySQL on Solaris using PKG distributions, see [Section 2.6, “Installing MySQL on Solaris”](#).

On Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris **tar** cannot handle long filenames. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU **tar** (**gtar**) to unpack the distribution. You can find a precompiled copy for Solaris at <http://dev.mysql.com/downloads/os-solaris.html>.

Sun native threads work only on Solaris 2.5 and higher. For Solaris 2.4 and

earlier, MySQL automatically uses MIT-pthreads. See [Section 2.9.5, “MIT-pthreads Notes”](#).

If you get the following error from **configure**, it means that you have something wrong with your compiler installation:

```
checking for restartable system calls... configure: error can not
run test programs while cross compiling
```

In this case, you should upgrade your compiler to a newer version. You may also be able to solve this problem by inserting the following row into the `config.cache` file:

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

If you are using Solaris on a SPARC, the recommended compiler is **gcc** 2.95.2 or 3.2. You can find this at <http://gcc.gnu.org/>. Note that **egcs** 1.1.1 and **gcc** 2.8.1 do not work reliably on SPARC.

The recommended **configure** line when using **gcc** 2.95.2 is:

```
CC=gcc CFLAGS="-O3" \  
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti \  
./configure --prefix=/usr/local/mysql --with-low-memory \  
--enable-assembler
```

If you have an UltraSPARC system, you can get 4% better performance by adding `-mcpu=v8 -wa, -xarch=v8plusa` to the `CFLAGS` and `CXXFLAGS` environment variables.

If you have Sun's Forte 5.0 (or newer) compiler, you can run **configure** like this:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \  
CXX=CC CXXFLAGS="-noex -mt" \  
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit binary with Sun's Forte compiler, use the following configuration options:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt -xarch=v9" \  
CXX=CC CXXFLAGS="-noex -mt -xarch=v9" ASFLAGS="-xarch=v9" \  
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit Solaris binary using **gcc**, add `-m64` to `CFLAGS` and `CXXFLAGS` and remove `--enable-asm-asm` from the **configure** line.

In the MySQL benchmarks, we obtained a 4% speed increase on UltraSPARC when using Forte 5.0 in 32-bit mode, as compared to using **gcc** 3.2 with the `-mcpu` flag.

If you create a 64-bit **mysqld** binary, it is 4% slower than the 32-bit binary, but can handle more threads and memory.

When using Solaris 10 for `x86_64`, you should mount any filesystems on which you intend to store InnoDB files with the `forcedirectio` option. (By default mounting is done without this option.) Failing to do so will cause a significant drop in performance when using the InnoDB storage engine on this platform.

If you get a problem with `fdatasync` or `sched_yield`, you can fix this by adding `LIBS=-lrt` to the **configure** line

For compilers older than WorkShop 5.3, you might have to edit the **configure** script. Change this line:

```
#if !defined(__STDC__) || __STDC__ != 1
```

To this:

```
#if !defined(__STDC__)
```

If you turn on `__STDC__` with the `-xc` option, the Sun compiler can't compile with the Solaris `pthread.h` header file. This is a Sun bug (broken compiler or broken include file).

If **mysqld** issues the following error message when you run it, you have tried to compile MySQL with the Sun compiler without enabling the `-mt` multi-thread option:

```
libc internal error: _rmutex_unlock: rmutex not held
```

Add `-mt` to `CFLAGS` and `CXXFLAGS` and recompile.

If you are using the SFW version of **gcc** (which comes with Solaris 8), you must add `/opt/sfw/lib` to the environment variable `LD_LIBRARY_PATH` before running

configure.

If you are using the **gcc** available from sunfreeware.com, you may have many problems. To avoid this, you should recompile **gcc** and GNU `binutils` on the machine where you are running them.

If you get the following error when compiling MySQL with **gcc**, it means that your **gcc** is not configured for your version of Solaris:

```
shell> gcc -O3 -g -O2 -DDEBUG_OFF -o thr_alarm ...
./thr_alarm.c: In function `signal_hand':
./thr_alarm.c:556: too many arguments to function `sigwait'
```

The proper thing to do in this case is to get the newest version of **gcc** and compile it with your current **gcc** compiler. At least for Solaris 2.5, almost all binary versions of **gcc** have old, unusable include files that break all programs that use threads, and possibly other programs as well.

Solaris does not provide static versions of all system libraries (`libpthreads` and `libdl`), so you cannot compile MySQL with `--static`. If you try to do so, you get one of the following errors:

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqlclient.so` to `/usr/lib`.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you have problems with **configure** trying to link with `-lz` when you don't have `zlib` installed, you have two options:

- If you want to be able to use the compressed communication protocol, you need to get and install `zlib` from `ftp.gnu.org`.
- Run **configure** with the `--with-named-z-libs=no` option when building MySQL.

If you are using **gcc** and have problems with loading user-defined functions (UDFs) into MySQL, try adding `-lgcc` to the link line for the UDF.

If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.

If too many processes try to connect very rapidly to **mysqld**, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

You might try starting the server with the `--back_log=50` option as a workaround for this. (Use `-o back_log=50` before MySQL 4.)

Solaris doesn't support core files for `setuid()` applications, so you can't get a core file from **mysqld** if you are using the `--user` option.

2.13.3.1. Solaris 2.7/2.8 Notes

Normally, you can use a Solaris 2.6 binary on Solaris 2.7 and 2.8. Most of the Solaris 2.6 issues also apply for Solaris 2.7 and 2.8.

MySQL should be able to detect new versions of Solaris automatically and enable workarounds for the following problems.

Solaris 2.7 / 2.8 has some bugs in the include files. You may see the following error when you use **gcc**:

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previ
definition
```

If this occurs, you can fix the problem by copying `/usr/include/widec.h` to `.../lib/gcc-lib/os/gcc-version/include` and changing line 41 from this:

```
#if !defined(lint) && !defined(__lint)
```

To this:

```
#if !defined(lint) && !defined(__lint) && !defined(getwc)
```

Alternatively, you can edit `/usr/include/widec.h` directly. Either way, after you make the fix, you should remove `config.cache` and run **configure** again.

If you get the following errors when you run **make**, it's because **configure** didn't detect the `curses.h` file (probably because of the error in `/usr/include/widec.h`):

```
In file included from mysql.cc:50:  
/usr/include/term.h:1060: syntax error before `,'  
/usr/include/term.h:1081: syntax error before `;'
```

The solution to this problem is to do one of the following:

- Configure with `CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H` and run `./configure`.
- Edit `/usr/include/widec.h` as indicated in the preceding discussion and re-run **configure**.
- Remove the `#define HAVE_TERM` line from the `config.h` file and run **make** again.

If your linker cannot find `-lz` when linking client programs, the problem is probably that your `libz.so` file is installed in `/usr/local/lib`. You can fix this problem by one of the following methods:

- Add `/usr/local/lib` to `LD_LIBRARY_PATH`.
- Add a link to `libz.so` from `/lib`.
- If you are using Solaris 8, you can install the optional `zlib` from your Solaris 8 CD distribution.

- Run **configure** with the `--with-named-z-libs=no` option when building MySQL.

2.13.3.2. Solaris x86 Notes

On Solaris 8 on x86, **mysqld** dumps core if you remove the debug symbols using `strip`.

If you are using **gcc** or **egcs** on Solaris x86 and you experience problems with core dumps under load, you should use the following **configure** command:

```
CC=gcc CFLAGS="-O3 -fomit-frame-pointer -DHAVE_CURSES_H" \  
CXX=gcc \  
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \  
-fno-exceptions -fno-rtti -DHAVE_CURSES_H" \  
./configure --prefix=/usr/local/mysql
```

This avoids problems with the `libstdc++` library and with C++ exceptions.

If this doesn't help, you should compile a debug version and run it with a trace file or under **gdb**. See [Section E.1.3, “Debugging **mysqld** under **gdb**”](#).

2.13.4. BSD Notes

This section provides information about using MySQL on variants of BSD Unix.

2.13.4.1. FreeBSD Notes

FreeBSD 4.x or newer is recommended for running MySQL, because the thread package is much more integrated. To get a secure and stable system, you should use only FreeBSD kernels that are marked `-RELEASE`.

The easiest (and preferred) way to install MySQL is to use the **mysql-server** and `mysql-client` ports available at <http://www.freebsd.org/>. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.

- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

It is recommended you use MIT-threads on FreeBSD 2.x, and native threads on FreeBSD 3 and up. It is possible to run with native threads on some late 2.2.x versions, but you may encounter problems shutting down **mysqld**.

Unfortunately, certain function calls on FreeBSD are not yet fully thread-safe. Most notably, this includes the `gethostbyname()` function, which is used by MySQL to convert hostnames into IP addresses. Under certain circumstances, the **mysqld** process suddenly causes 100% CPU load and is unresponsive. If you encounter this problem, try to start MySQL using the `--skip-name-resolve` option.

Alternatively, you can link MySQL on FreeBSD 4.x against the LinuxThreads library, which avoids a few of the problems that the native FreeBSD thread implementation has. For a very good comparison of LinuxThreads versus native threads, see Jeremy Zawodny's article *FreeBSD or Linux for your MySQL Server?* at <http://jeremy.zawodny.com/blog/archives/000697.html>.

Known problem when using LinuxThreads on FreeBSD is:

- The connection times (`wait_timeout`, `interactive_timeout` and `net_read_timeout`) values are not honored. The symptom is that persistent connections can hang for a very long time without getting closed down and that a 'kill' for a thread will not take affect until the thread does it a new command

This is probably a signal handling problem in the thread library where the signal doesn't break a pending read. This is supposed to be fixed in FreeBSD 5.0

The MySQL build process requires GNU make (**gmake**) to work. If GNU **make** is not available, you must install it first before compiling MySQL.

The recommended way to compile and install MySQL on FreeBSD with **gcc**

(2.95.2 and up) is:

```
CC=gcc CFLAGS="-O2 -fno-strength-reduce" \  
  CXX=gcc CXXFLAGS="-O2 -fno-rtti -fno-exceptions \  
  -felide-constructors -fno-strength-reduce" \  
  ./configure --prefix=/usr/local/mysql --enable-asm  
gmake  
gmake install  
cd /usr/local/mysql  
bin/mysql_install_db --user=mysql  
bin/mysqld_safe &
```

If you notice that **configure** uses MIT-pthreads, you should read the MIT-pthreads notes. See [Section 2.9.5, “MIT-pthreads Notes”](#).

If you get an error from **make install** that it can't find `/usr/include/pthreads`, **configure** didn't detect that you need MIT-pthreads. To fix this problem, remove `config.cache`, and then re-run **configure** with the `--with-mit-threads` option.

Be sure that your name resolver setup is correct. Otherwise, you may experience resolver delays or failures when connecting to **mysqld**. Also make sure that the `localhost` entry in the `/etc/hosts` file is correct. The file should start with a line similar to this:

```
127.0.0.1      localhost localhost.your.domain
```

FreeBSD is known to have a very low default file handle limit. See [Section A.2.17, “File Not Found”](#). Start the server by using the `--open-files-limit` option for **mysqld_safe**, or raise the limits for the **mysqld** user in `/etc/login.conf` and rebuild it with `cap_mkdb /etc/login.conf`. Also be sure that you set the appropriate class for this user in the password file if you are not using the default (use `chpass mysqld-user-name`). See [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#).

FreeBSD limits the size of a process to 512MB, even if you have much more RAM available on the system. So you may get an error such as this:

```
Out of memory (Needed 16391 bytes)
```

In current versions of FreeBSD (at least 4.x and greater), you may increase this limit by adding the following entries to the `/boot/loader.conf` file and rebooting the machine (these are not settings that can be changed at run time

with the **sysctl** command):

```
kern.maxdsiz="1073741824" # 1GB
kern.dfldsiz="1073741824" # 1GB
kern.maxssiz="134217728" # 128MB
```

For older versions of FreeBSD, you must recompile your kernel to change the maximum data segment size for a process. In this case, you should look at the `MAXDSIZ` option in the `LINT` config file for more information.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See [Appendix F, Environment Variables](#).

2.13.4.2. NetBSD Notes

To compile on NetBSD, you need GNU **make**. Otherwise, the build process fails when **make** tries to run `lint` on C++ files.

2.13.4.3. OpenBSD 2.5 Notes

On OpenBSD 2.5, you can compile MySQL with native threads with the following options:

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.13.4.4. BSD/OS Version 2.x Notes

If you get the following error when compiling MySQL, your **ulimit** value for virtual memory is too low:

```
item_func.h: In method
`Item_func_ge::Item_func_ge(const Item_func_ge &)`:
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

Try using **ulimit -v 80000** and run **make** again. If this doesn't work and you are using **bash**, try switching to **csh** or **sh**; some BSDI users have reported problems with **bash** and **ulimit**.

If you are using **gcc**, you may also use have to use the `--with-low-memory` flag

for **configure** to be able to compile `sql_yacc.cc`.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See [Appendix F, *Environment Variables*](#).

2.13.4.5. BSD/OS Version 3.x Notes

Upgrade to BSD/OS 3.1. If that is not possible, install BSDIpatch M300-038.

Use the following command when configuring MySQL:

```
env CXX=shlcc++ CC=shlcc2 \  
./configure \  
  --prefix=/usr/local/mysql \  
  --localstatedir=/var/mysql \  
  --without-perl \  
  --with-unix-socket-path=/var/mysql/mysql.sock
```

The following is also known to work:

```
env CC=gcc CXX=gcc CXXFLAGS=-O3 \  
./configure \  
  --prefix=/usr/local/mysql \  
  --with-unix-socket-path=/var/mysql/mysql.sock
```

You can change the directory locations if you wish, or just use the defaults by not specifying any locations.

If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to **mysqld**. This runs all threads with the same priority. On BSDI 3.1, this gives better performance, at least until BSDI fixes its thread scheduler.

If you get the error `virtual memory exhausted` while compiling, you should try using **ulimit -v 80000** and running **make** again. If this doesn't work and you are using **bash**, try switching to **csh** or **sh**; some BSDI users have reported problems with **bash** and **ulimit**.

2.13.4.6. BSD/OS Version 4.x Notes

BSDI 4.x has some thread-related bugs. If you want to use MySQL on this, you

should install all thread-related patches. At least M400-023 should be installed.

On some BSDI 4.x systems, you may get problems with shared libraries. The symptom is that you can't execute any client programs, for example, **mysqladmin**. In this case, you need to reconfigure not to use shared libraries with the `--disable-shared` option to configure.

Some customers have had problems on BSDI 4.0.1 that the **mysqld** binary after a while can't open tables. This occurs because some library/system-related bug causes **mysqld** to change current directory without having asked for that to happen.

The fix is to either upgrade MySQL to at least version 3.23.34 or, after running **configure**, remove the line `#define HAVE_REALPATH` from `config.h` before running **make**.

Note that this means that you can't symbolically link a database directories to another database directory or symbolic link a table to another database on BSDI. (Making a symbolic link to another disk is okay).

2.13.5. Other Unix Notes

2.13.5.1. HP-UX Version 10.20 Notes

There are a couple of small problems when compiling MySQL on HP-UX. We recommend that you use **gcc** instead of the HP-UX native compiler, because **gcc** produces better code.

We recommend using **gcc** 2.95 on HP-UX. Don't use high optimization flags (such as `-O6`) because they may not be safe on HP-UX.

The following **configure** line should work with **gcc** 2.95:

```
CFLAGS="-I/opt/dce/include -fpic" \  
CXXFLAGS="-I/opt/dce/include -felide-constructors -fno-exceptions \  
-fno-rtti" \  
CXX=gcc \  
./configure --with-pthread \  
    --with-named-thread-libs='-ldce' \  
    --prefix=/usr/local/mysql --disable-shared
```

The following **configure** line should work with **gcc 3.1**:

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc \  
CXXFLAGS="-DHPUX -I/opt/dce/include -felide-constructors \  
-fno-exceptions -fno-rtti -O3 -fPIC" \  
./configure --prefix=/usr/local/mysql \  
--with-extra-charsets=complex --enable-thread-safe-client \  
--enable-local-infile --with-pthread \  
--with-named-thread-libs=-ldce --with-lib-ccflags=-fPIC \  
--disable-shared
```

2.13.5.2. HP-UX Version 11.x Notes

Because of some critical bugs in the standard HP-UX libraries, you should install the following patches before trying to run MySQL on HP-UX 11.0:

```
PHKL_22840 Streams cumulative  
PHNE_22397 ARPA cumulative
```

This solves the problem of getting EWOULDBLOCK from `recv()` and EBADF from `accept()` in threaded applications.

If you are using **gcc 2.95.1** on an unpatched HP-UX 11.x system, you may get the following error:

```
In file included from /usr/include/unistd.h:11,  
                 from ../include/global.h:125,  
                 from mysql_priv.h:15,  
                 from item.cc:19:  
/usr/include/sys/unistd.h:184: declaration of C function ...  
/usr/include/sys/pthread.h:440: previous declaration ...  
In file included from item.h:306,  
                 from mysql_priv.h:158,  
                 from item.cc:19:
```

The problem is that HP-UX does not define `pthread_atfork()` consistently. It has conflicting prototypes in `/usr/include/sys/unistd.h:184` and `/usr/include/sys/pthread.h:440`.

One solution is to copy `/usr/include/sys/unistd.h` into `mysql/include` and edit `unistd.h` and change it to match the definition in `pthread.h`. Look for this line:

```
extern int pthread_atfork(void (*prepare)(), void (*parent)(),
```

```
void (*child)());
```

Change it to look like this:

```
extern int pthread_atfork(void (*prepare)(void), void (*parent)(void),  
                          void (*child)(void));
```

After making the change, the following **configure** line should work:

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \  
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \  
./configure --prefix=/usr/local/mysql --disable-shared
```

If you are using HP-UX compiler, you can use the following command (which has been tested with **cc B.11.11.04**):

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \  
--with-extra-character-set=complex
```

You can ignore any errors of the following type:

```
aCC: warning 901: unknown option: ` -3': use +help for online  
documentation
```

If you get the following error from **configure**, verify that you don't have the path to the K&R compiler before the path to the HP-UX C and C++ compiler:

```
checking for cc option to accept ANSI C... no  
configure: error: MySQL requires an ANSI C compiler (and a C++ compi  
Try gcc. See the Installation chapter in the Reference Manual.
```

Another reason for not being able to compile is that you didn't define the +DD64 flags as just described.

Another possibility for HP-UX 11 is to use the MySQL binaries provided at <http://dev.mysql.com/downloads/>, which we have built and tested ourselves. We have also received reports that the HP-UX 10.20 binaries supplied by MySQL can be run successfully on HP-UX 11. If you encounter problems, you should be sure to check your HP-UX patch level.

2.13.5.3. IBM-AIX notes

Automatic detection of x1C is missing from Autoconf, so a number of variables

need to be set before running **configure**. The following example uses the IBM compiler:

```
export CC="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192 "  
export CXX="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192 "  
export CFLAGS="-I /usr/local/include"  
export LDFLAGS="-L /usr/local/lib"  
export CPPFLAGS=$CFLAGS  
export CXXFLAGS=$CFLAGS
```

```
./configure --prefix=/usr/local \  
            --localstatedir=/var/mysql \  
            --sbindir='/usr/local/bin' \  
            --libexecdir='/usr/local/bin' \  
            --enable-thread-safe-client \  
            --enable-large-files
```

The preceding options are used to compile the MySQL distribution that can be found at <http://www-frec.bull.com/>.

If you change the `-O3` to `-O2` in the preceding **configure** line, you must also remove the `-qstrict` option. This is a limitation in the IBM C compiler.

If you are using **gcc** or **egcs** to compile MySQL, you *must* use the `-fno-exceptions` flag, because the exception handling in **gcc/egcs** is not thread-safe! (This is tested with **egcs** 1.1.) There are also some known problems with IBM's assembler that may cause it to generate bad code when used with **gcc**.

We recommend the following **configure** line with **egcs** and **gcc** 2.95 on AIX:

```
CC="gcc -pipe -mcpu=power -wa, -many" \  
CXX="gcc -pipe -mcpu=power -wa, -many" \  
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \  
./configure --prefix=/usr/local/mysql --with-low-memory
```

The `-wa, -many` option is necessary for the compile to be successful. IBM is aware of this problem but is in no hurry to fix it because of the workaround that is available. We don't know if the `-fno-exceptions` is required with **gcc** 2.95, but because MySQL doesn't use exceptions and the option generates faster code, we recommend that you should always use it with **egcs** / **gcc**.

If you get a problem with assembler code, try changing the `-mcpu=xxx` option to match your CPU. Typically `power2`, `power`, or `powerpc` may need to be used.

Alternatively, you might need to use 604 or 604e. We are not positive but suspect that power would likely be safe most of the time, even on a power2 machine.

If you don't know what your CPU is, execute a `uname -m` command. It produces a string that looks like `000514676700`, with a format of `xyyyyyymmss` where `xx` and `ss` are always `00`, `yyyyyy` is a unique system ID and `mm` is the ID of the CPU Planar. A chart of these values can be found at http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/uname.htm.

This gives you a machine type and a machine model you can use to determine what type of CPU you have.

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring as follows:

```
CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \  
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti \  
-DDONT_USE_THR_ALARM" \  
./configure --prefix=/usr/local/mysql --with-debug \  
--with-low-memory
```

This doesn't affect the performance of MySQL, but has the side effect that you can't kill clients that are "sleeping" on a connection with **mysqladmin kill** or **mysqladmin shutdown**. Instead, the client dies when it issues its next command.

On some versions of AIX, linking with `libbind.a` makes `getservbyname()` dump core. This is an AIX bug and should be reported to IBM.

For AIX 4.2.1 and **gcc**, you have to make the following changes.

After configuring, edit `config.h` and `include/my_config.h` and change the line that says this:

```
#define HAVE_SNPRINTF 1
```

to this:

```
#undef HAVE_SNPRINTF
```

And finally, in `mysqld.cc`, you need to add a prototype for `initgroups()`.

```
#ifndef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

If you need to allocate a lot of memory to the **mysqld** process, it's not enough to just use **ulimit -d unlimited**. You may also have to modify **mysqld_safe** to add a line something like this:

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

You can find more information about using a lot of memory at http://publib16.boulder.ibm.com/pseries/en_US/aixprgpd/genprog/lrg_prg_supp

Users of AIX 4.3 should use **gmake** instead of the **make** utility included with AIX.

As of AIX 4.1, the C compiler has been unbundled from AIX as a separate product. We recommend using **gcc 3.3.2**, which can be obtained here: <ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/RPMS/ppc/gcc/>

The steps for compiling MySQL on AIX with **gcc 3.3.2** are similar to those for using **gcc 2.95** (in particular, the need to edit **config.h** and **my_config.h** after running **configure**). However, before running **configure**, you should also patch the **curses.h** file as follows:

```
/opt/freeware/lib/gcc-lib/powerpc-ibm-aix5.2.0.0/3.3.2/include/curses.h
Mon Dec 26 02:17:28 2005
--- /opt/freeware/lib/gcc-lib/powerpc-ibm-aix5.2.0.0/3.3.2/include/curses.h
Mon Dec 26 02:40:13 2005
*****
*** 2023,2029 ***
```

```

#endif /* _AIX32_CURSES */
! #if defined(__USE_FIXED_PROTOTYPES__) || defined(__cplusplus) || d
(__STRICT_ANSI__)
extern int delwin (WINDOW *);
extern int endwin (void);
extern int getcurx (WINDOW *);
--- 2023,2029 ----
```

```

#endif /* _AIX32_CURSES */
! #if 0 && (defined(__USE_FIXED_PROTOTYPES__) || defined(__cplusplus)
|| defined
```

```
(__STRICT_ANSI__)\n  extern int delwin (WINDOW *);\n  extern int endwin (void);\n  extern int getcurx (WINDOW *);
```

2.13.5.4. SunOS 4 Notes

On SunOS 4, MIT-pthreads is needed to compile MySQL. This in turn means you need GNU **make**.

Some SunOS 4 systems have problems with dynamic libraries and **libtool**. You can use the following **configure** line to avoid this problem:

```
./configure --disable-shared --with-mysqld-ldflags=-all-static
```

When compiling `readline`, you may get warnings about duplicate defines. These can be ignored.

When compiling **mysqld**, there are some implicit declaration of function warnings. These can be ignored.

2.13.5.5. Alpha-DEC-UNIX Notes (Tru64)

If you are using **egcs** 1.1.2 on Digital Unix, you should upgrade to **gcc** 2.95.2, because **egcs** on DEC has some serious bugs!

When compiling threaded programs under Digital Unix, the documentation recommends using the `-pthread` option for **cc** and **cxx** and the `-lmach -lexc` libraries (in addition to `-lpthread`). You should run **configure** something like this:

```
CC="cc -pthread" CXX="cxx -pthread -O" \  
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

When compiling **mysqld**, you may see a couple of warnings like this:

```
mysqld.cc: In function void handle_connections():  
mysqld.cc:626: passing long unsigned int '*' as argument 3 of  
accept(int,sockaddr *, int *)'
```

You can safely ignore these warnings. They occur because **configure** can detect

only errors, not warnings.

If you start the server directly from the command line, you may have problems with it dying when you log out. (When you log out, your outstanding processes receive a SIGHUP signal.) If so, try starting the server like this:

```
nohup mysqld [options] &
```

nohup causes the command following it to ignore any SIGHUP signal sent from the terminal. Alternatively, start the server by running **mysqld_safe**, which invokes **mysqld** using **nohup** for you. See [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#).

If you get a problem when compiling `mysys/get_opt.c`, just remove the `#define _NO_PROTO` line from the start of that file.

If you are using Compaq's CC compiler, the following **configure** line should work:

```
CC="cc -pthread"
CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \
        -speculate all -arch host"
CXX="cxx -pthread"
CXXFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed \
        -speculate all -arch host -noexceptions -nortti"
export CC CFLAGS CXX CXXFLAGS
./configure \
    --prefix=/usr/local/mysql \
    --with-low-memory \
    --enable-large-files \
    --enable-shared=yes \
    --with-named-thread-libs="-lpthread -lmach -lexc -lc"
gnumake
```

If you get a problem with **libtool** when compiling with shared libraries as just shown, when linking **mysql**, you should be able to get around this by issuing these commands:

```
cd mysql
/bin/sh ../libtool --mode=link cxx -pthread -O3 -DDEBUG_OFF \
    -O4 -ansi_alias -ansi_args -fast -inline speed \
    -speculate all \ -arch host -DUNDEF_HAVE_GETHOSTBYNAME_R \
    -o mysql mysql.o readline.o sql_string.o completion_hash.o \
    ../readline/libreadline.a -lcurses \
```

```
    ../libmysql/.libs/libmysqlclient.so  -lm
cd ..
gnumake
gnumake install
scripts/mysql_install_db
```

2.13.5.6. Alpha-DEC-OSF/1 Notes

If you have problems compiling and have DEC **CC** and **gcc** installed, try running **configure** like this:

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \  
./configure --prefix=/usr/local/mysql
```

If you get problems with the `c_asm.h` file, you can create and use a 'dummy' `c_asm.h` file with:

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \  
CXX=gcc CXXFLAGS=-O3 \  
./configure --prefix=/usr/local/mysql
```

Note that the following problems with the **ld** program can be fixed by downloading the latest DEC (Compaq) patch kit from:

<http://ftp.support.compaq.com/public/unix/>.

On OSF/1 V4.0D and compiler "DEC C V5.6-071 on Digital Unix V4.0 (Rev. 878)," the compiler had some strange behavior (undefined asm symbols). `/bin/ld` also appears to be broken (problems with `_exit` undefined errors occurring while linking **mysqld**). On this system, we have managed to compile MySQL with the following **configure** line, after replacing `/bin/ld` with the version from OSF 4.0C:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

With the Digital compiler "C++ V6.1-029," the following should work:

```
CC=cc -pthread
CFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \  
        -speculate all -arch host
CXX=cxx -pthread
CXXFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \  
        -speculate all -arch host -noexceptions -nortti
```

```
export CC CFLAGS CXX CXXFLAGS
./configure --prefix=/usr/mysql/mysql \
            --with-mysqld-ldflags=-all-static --disable-shared \
            --with-named-thread-libs="-lmach -lexc -lc"
```

In some versions of OSF/1, the `alloca()` function is broken. Fix this by removing the line in `config.h` that defines `'HAVE_ALLOCA'`.

The `alloca()` function also may have an incorrect prototype in `/usr/include/alloca.h`. This warning resulting from this can be ignored.

configure uses the following thread libraries automatically: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

When using **gcc**, you can also try running **configure** like this:

```
CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ...
```

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring with:

```
CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure ...
```

This does not affect the performance of MySQL, but has the side effect that you can't kill clients that are "sleeping" on a connection with **mysqladmin kill** or **mysqladmin shutdown**. Instead, the client dies when it issues its next command.

With **gcc** 2.95.2, you may encounter the following compile error:

```
sql_acl.cc:1456: Internal compiler error in `scan_region',
at except.c:2566
Please submit a full bug report.
```

To fix this, you should change to the `sql` directory and do a cut-and-paste of the last **gcc** line, but change `-O3` to `-O0` (or add `-O0` immediately after **gcc** if you don't have any `-O` option on your compile line). After this is done, you can just change back to the top-level directory and run **make** again.

2.13.5.7. SGI Irix Notes

If you are using Irix 6.5.3 or newer, **mysqld** is able to create threads only if you run it as a user that has CAP_SCHED_MGT privileges (such as root) or give the **mysqld** server this privilege with the following shell command:

```
chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

You may have to undefine some symbols in config.h after running **configure** and before compiling.

In some Irix implementations, the `alloca()` function is broken. If the **mysqld** server dies on some SELECT statements, remove the lines from config.h that define HAVE_ALLOC and HAVE_ALLOCA_H. If **mysqladmin create** doesn't work, remove the line from config.h that defines HAVE_READDIR_R. You may have to remove the HAVE_TERM_H line as well.

SGI recommends that you install all the patches on this page as a set:
http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

At the very minimum, you should install the latest kernel rollup, the latest r1d rollup, and the latest libc rollup.

You definitely need all the POSIX patches on this page, for pthreads support:

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

If you get the something like the following error when compiling mysql.cc:

```
"/usr/include/curses.h", line 82: error(1084):  
invalid combination of type
```

Type the following in the top-level directory of your MySQL source tree:

```
extra/replace bool curses_bool < /usr/include/curses.h > include/cur  
make
```

There have also been reports of scheduling problems. If only one thread is running, performance is slow. Avoid this by starting another client. This may lead to a two-to-tenfold increase in execution speed thereafter for the other thread. This is a poorly understood problem with Irix threads; you may have to

improvise to find solutions until this can be fixed.

If you are compiling with **gcc**, you can use the following **configure** command:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \  
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \  
--with-named-thread-libs=-lpthread
```

On Irix 6.5.11 with native Irix C and C++ compilers ver. 7.3.1.2, the following is reported to work

```
CC=cc CXX=CC CFLAGS='-O3 -n32 -TARG:platform=IP22 -I/usr/local/inclu-  
-L/usr/local/lib' CXXFLAGS='-O3 -n32 -TARG:platform=IP22 \  
-I/usr/local/include -L/usr/local/lib' \  
./configure --prefix=/usr/local/mysql --with-innodb --with-berkeley-  
--with-libwrap=/usr/local \  
--with-named-curses-libs=/usr/local/lib/libncurses.a
```

2.13.5.8. SCO UNIX and OpenServer 5.0.x Notes

The current port is tested only on sco3.2v5.0.5, sco3.2v5.0.6, and sco3.2v5.0.7 systems. There has also been progress on a port to sco3.2v4.2. Open Server 5.0.8 (Legend) has native threads and allows files greater than 2GB. The current maximum file size is 2GB.

We have been able to compile MySQL with the following **configure** command on OpenServer with **gcc** 2.95.3.

```
CC=gcc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \  
CXX=gcc CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \  
./configure --prefix=/usr/local/mysql \  
--enable-thread-safe-client --with-innodb \  
--with-openssl --with-vio --with-extra-charsets=complex
```

gcc is available at <ftp://ftp.sco.com/pub/openserver5/opensrc/gnutools-5.0.7Kj>.

This development system requires the OpenServer Execution Environment Supplement oss646B on OpenServer 5.0.6 and oss656B and The OpenSource libraries found in gwlibs. All OpenSource tools are in the opensrc directory. They are available at <ftp://ftp.sco.com/pub/openserver5/opensrc/>.

We recommend using the latest production release of MySQL.

SCO provides operating system patches at <ftp://ftp.sco.com/pub/openserver5> for OpenServer 5.0.[0-6] and <ftp://ftp.sco.com/pub/openserverv5/507> for OpenServer 5.0.7.

SCO provides information about security fixes at <ftp://ftp.sco.com/pub/security/OpenServer> for OpenServer 5.0.x.

The maximum file size on an OpenServer 5.0.x system is 2GB.

The total memory which can be allocated for streams buffers, clists, and lock records cannot exceed 60MB on OpenServer 5.0.x.

Streams buffers are allocated in units of 4096 byte pages, clists are 70 bytes each, and lock records are 64 bytes each, so:

$$(NSTRPAGES \times 4096) + (NCLIST \times 70) + (MAX_FLCKREC \times 64) \leq 62914560$$

Follow this procedure to configure the Database Services option. If you are unsure whether an application requires this, see the documentation provided with the application.

1. Log in as root.
2. Enable the SUDS driver by editing the `/etc/conf/sdevice.d/suds` file. Change the `N` in the second field to a `Y`.
3. Use **mkdev aio** or the Hardware/Kernel Manager to enable support for asynchronous I/O and relink the kernel. To allow users to lock down memory for use with this type of I/O, update the `aiomemlock(F)` file. This file should be updated to include the names of users that can use AIO and the maximum amounts of memory they can lock down.
4. Many applications use `setuid` binaries so that you need to specify only a single user. See the documentation provided with the application to determine whether this is the case for your application.

After you complete this process, reboot the system to create a new kernel incorporating these changes.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

Value	Default	Min	Max
-----	-----	---	---
NBUF	0	24	450000
NHBUF	0	32	524288
NMPBUF	0	12	512
MAX_INODE	0	100	64000
MAX_FILE	0	100	64000
CTBUFSIZE	128	0	256
MAX_PROC	0	50	16000
MAX_REGION	0	500	160000
NCLIST	170	120	16640
MAXUP	100	15	16000
NOFILES	110	60	11000
NHINODE	128	64	8192
NAUTOUP	10	0	60
NGROUPS	8	0	128
BDFLUSHR	30	1	300
MAX_FLCKREC	0	50	16000
PUTBUFSZ	8000	2000	20000
MAXSLICE	100	25	100
ULIMIT	4194303	2048	4194303
* Streams Parameters			
NSTREAM	64	1	32768
NSTRPUSH	9	9	9
NMUXLINK	192	1	4096
STRMSGSZ	16384	4096	524288
STRCTLSZ	1024	1024	1024
STRMAXBLK	524288	4096	524288
NSTRPAGES	500	0	8000
STRSPLITFRAC	80	50	100
NLOG	3	3	3
NUMSP	64	1	256
NUMTIM	16	1	8192
NUMTRW	16	1	8192
* Semaphore Parameters			
SEMAP	10	10	8192
SEMMNI	10	10	8192
SEMMNS	60	60	8192
SEMMNU	30	10	8192
SEMMSL	25	25	150
SEMOPM	10	10	1024
SEMUME	10	10	25
SEVMX	32767	32767	32767
SEMAEM	16384	16384	16384
* Shared Memory Parameters			
SHMMAX	524288	131072	2147483647
SHMMIN	1	1	1
SHMMNI	100	100	2000
FILE	0	100	64000
NMOUNT	0	4	256

NPROC	0	50	16000
NREGION	0	500	160000

We recommend setting these values as follows:

- NOFILES should be 4096 or 2048.
- MAXUP should be 2048.

To make changes to the kernel, use the **idtune name parameter** command. **idtune** modifies the `/etc/conf/cf.d/stune` file for you. For example, to change SEMMS to 200, execute this command as root:

```
# /etc/conf/bin/idtune SEMMNS 200
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

We recommend tuning the system, but the proper parameter values to use depend on the number of users accessing the application or database and size of the database (that is, the used buffer pool). The following kernel parameters can be set with **idtune**:

- SHMMAX (recommended setting: 128MB) and SHMSEG (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- NOFILES and MAXUP should be set to at least 2048.
- MAXPROC should be set to at least 3000/4000 (depends on number of users) or more.
- We also recommend using the following formulas to calculate values for SEMMSL, SEMMNS, and SEMMNU:

```
SEMMSL = 13
```

13 is what has been found to be the best for both Progress and MySQL.

```
SEMMNS = SEMMSL × number of db servers to be run on the system
```

Set SEMMNS to the value of SEMMSL multiplied by the number of database servers (maximum) that you are running on the system at one time.

SEMMNU = SEMMNS

Set the value of SEMMNU to equal the value of SEMMNS. You could probably set this to 75% of SEMMNS, but this is a conservative estimate.

You need to at least install the SCO OpenServer Linker and Application Development Libraries or the OpenServer Development System to use **gcc**. You cannot use the GCC Dev system without installing one of these.

You should get the FSU Pthreads package and install it first. This can be found at <http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz>. You can also get a precompiled package from <ftp://ftp.zenez.com/pub/zenez/prgms/FSU-threads-3.14.tar.gz>.

FSU Pthreads can be compiled with SCO Unix 4.2 with tcpip, or using OpenServer 3.0 or Open Desktop 3.0 (OS 3.0 ODT 3.0) with the SCO Development System installed using a good port of GCC 2.5.x. For ODT or OS 3.0, you need a good port of GCC 2.5.x. There are a lot of problems without a good port. The port for this product requires the SCO Unix Development system. Without it, you are missing the libraries and the linker that is needed. You also need `SCO-3.2v4.2-includes.tar.gz`. This file contains the changes to the SCO Development include files that are needed to get MySQL to build. You need to replace the existing system include files with these modified header files. They can be obtained from <ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz>.

To build FSU Pthreads on your system, all you should need to do is run GNU **make**. The `Makefile` in `FSU-threads-3.14.tar.gz` is set up to make FSU-threads.

You can run `./configure` in the `threads/src` directory and select the SCO OpenServer option. This command copies `Makefile.SCO5` to `Makefile`. Then run **make**.

To install in the default `/usr/include` directory, log in as root, and then `cd` to the `thread/src` directory and run **make install**.

Remember that you must use GNU **make** to build MySQL.

Note: If you don't start **mysqld_safe** as root, you should get only the default 110 open files per process. **mysqld** writes a note about this in the log file.

With SCO 3.2V4.2, you should use FSU Pthreads version 3.14 or newer. The following **configure** command should work:

```
CFLAGS="-D_XOPEN_XPG4" CXX=gcc CXXFLAGS="-D_XOPEN_XPG4" \  
./configure \  
  --prefix=/usr/local/mysql \  
  --with-named-thread-libs="-lgthreads -lsocket -lgen -lgthreads" \  
  --with-named-curses-libs="-lcurses"
```

You may have problems with some include files. In this case, you can find new SCO-specific include files at [ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz](http://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz).

You should unpack this file in the `include` directory of your MySQL source tree.

SCO development notes:

- MySQL should automatically detect FSU Pthreads and link **mysqld** with `-lgthreads -lsocket -lgthreads`.
- The SCO development libraries are re-entrant in FSU Pthreads. SCO claims that its library functions are re-entrant, so they must be re-entrant with FSU Pthreads. FSU Pthreads on OpenServer tries to use the SCO scheme to make re-entrant libraries.
- FSU Pthreads (at least the version at [ftp://ftp.zenez.com](http://ftp.zenez.com)) comes linked with GNU `malloc`. If you encounter problems with memory usage, make sure that `gmalloc.o` is included in `libgthreads.a` and `libgthreads.so`.
- In FSU Pthreads, the following system calls are pthreads-aware: `read()`, `write()`, `getmsg()`, `connect()`, `accept()`, `select()`, and `wait()`.
- The CSSA-2001-SCO.35.2 (the patch is listed in custom as `erg711905-dscr_remap` security patch (version 2.0.0)) breaks FSU threads and makes **mysqld** unstable. You have to remove this one if you want to run **mysqld** on an OpenServer 5.0.6 machine.
- If you use SCO OpenServer 5, you may need to recompile FSU pthreads

with `-DDRAFT7` in `CFLAGS`. Otherwise, InnoDB may hang at a **mysqld** startup.

- SCO provides operating system patches at <ftp://ftp.sco.com/pub/openserver5> for OpenServer 5.0.x.
- SCO provides security fixes and `libsocket.so.2` at <ftp://ftp.sco.com/pub/security/OpenServer> and <ftp://ftp.sco.com/pub/security/sse> for OpenServer 5.0.x.
- Pre-OSR506 security fixes. Also, the `telnetd` fix at <ftp://stage.caldera.com/pub/security/openserver/> or <ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/> as both `libsocket.so.2` and `libresolv.so.1` with instructions for installing on pre-OSR506 systems.

It's probably a good idea to install these patches before trying to compile/use MySQL.

Beginning with Legend/OpenServer 6.0.0, there are native threads and no 2GB file size limit.

2.13.5.9. SCO OpenServer 6.0.x Notes

OpenServer 6 includes these key improvements:

- Larger file support up to 1 TB
- Multiprocessor support increased from 4 to 32 processors
- Increased memory support up to 64GB
- Extending the power of UnixWare into OpenServer 6
- Dramatic performance improvement

OpenServer 6.0.0 commands are organized as follows:

- `/bin` is for commands that behave exactly the same as on OpenServer 5.0.x.
- `/u95/bin` is for commands that have better standards conformance, for

example Large File System (LFS) support.

- `/udk/bin` is for commands that behave the same as on UnixWare 7.1.4. The default is for the LFS support.

The following is a guide to setting `PATH` on OpenServer 6. If the user wants the traditional OpenServer 5.0.x then `PATH` should be `/bin` first. If the user wants LFS support, the path should be `/u95/bin:/bin`. If the user wants UnixWare 7 support first, the path would be `/udk/bin:/u95/bin:/bin:`.

We recommend using the latest production release of MySQL. Should you choose to use an older release of MySQL on OpenServer 6.0.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

MySQL distribution files with names of the following form are **tar** archives of media are tar archives of media images suitable for installation with the SCO Software Manager (`/etc/custom`) on SCO OpenServer 6:

```
mysql-PRODUCT-5.0.25-sco-osr6-i686.VOLS.tar
```

A distribution where *PRODUCT* is *pro-cert* is the Commercially licensed MySQL Pro Certified server. A distribution where *PRODUCT* is *pro-gpl-cert* is the MySQL Pro Certified server licensed under the terms of the General Public License (GPL).

Select whichever distribution you wish to install and, after download, extract the **tar** archive into an empty directory. For example:

```
shell> mkdir /tmp/mysql-pro
shell> cd /tmp/mysql-pro
shell> tar xf /tmp/mysql-pro-cert-5.0.25-sco-osr6-i686.VOLS.tar
```

Prior to installation, back up your data in accordance with the procedures outlined in [Section 2.11, “Upgrading MySQL”](#).

Remove any previously installed **pkgadd** version of MySQL:

```
shell> pkginfo mysql 2>&1 > /dev/null && pkgrm mysql
```

Install MySQL Pro from media images using the SCO Software Manager:

```
shell> /etc/custom -p SCO:MySQL -i -z /tmp/mysql-pro
```

Alternatively, the SCO Software Manager can be displayed graphically by clicking on the Software Manager icon on the desktop, selecting Software -> Install New, selecting the host, selecting Media Images for the Media Device, and entering /tmp/mysql-pro as the Image Directory.

After installation, run **mkdev mysql** as the root user to configure your newly installed MySQL Pro Certified server.

Note: The installation procedure for VOLS packages does not create the mysql user and group that the package uses by default. You should either create the mysql user and group, or else select a different user and group using an option in **mkdev mysql**.

If you wish to configure your MySQL Pro server to interface with the Apache Web server via PHP, download and install the PHP update from SCO at <ftp://ftp.sco.com/pub/updates/OpenServer/SCOSA-2006.17/>.

We have been able to compile MySQL with the following **configure** command on OpenServer 6.0.x:

```
CC=cc CFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \  
CXX=CC CXXFLAGS="-D_FILE_OFFSET_BITS=64 -O3" \  
./configure --prefix=/usr/local/mysql \  
    --enable-thread-safe-client --with-berkeley-db \  
    --with-extra-charsets=complex \  
    --build=i686-unknown-sysv5SCO_SV6.0.0
```

If you use **gcc**, you must use **gcc 2.95.3** or newer.

```
CC=gcc CXX=g++ ... ./configure ...
```

The version of Berkeley DB that comes with either UnixWare 7.1.4 or OpenServer 6.0.0 is not used when building MySQL. MySQL instead uses its own version of Berkeley DB. The **configure** command needs to build both a static and a dynamic library in `src_directory/bdb/build_unix/`, but it does not with MySQL's own BDB version. The workaround is as follows.

1. Configure as normal for MySQL.
2. `cd bdb/build_unix/`

3. `cp -p Makefile Makefile.sav`
4. Use same options and run `../dist/configure`.
5. Run **gmake**.
6. `cp -p Makefile.sav Makefile`
7. Change location to the top source directory and run **gmake**.

This allows both the shared and dynamic libraries to be made and work.

SCO provides OpenServer 6 operating system patches at <ftp://ftp.sco.com/pub/openserver6>.

SCO provides information about security fixes at <ftp://ftp.sco.com/pub/security/OpenServer>.

By default, the maximum file size on a OpenServer 6.0.0 system is 1TB. Some operating system utilities have a limitation of 2GB. The maximum possible file size on UnixWare 7 is 1TB with VXFS or HTFS.

OpenServer 6 can be configured for large file support (file sizes greater than 2GB) by tuning the UNIX kernel.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

Value	Default	Min	Max
-----	-----	---	---
SVMMLIM	0x9000000	0x1000000	0x7FFFFFFF
HVMMLIM	0x9000000	0x1000000	0x7FFFFFFF

To make changes to the kernel, use the **idtune name parameter** command. **idtune** modifies the `/etc/conf/cf.d/stune` file for you. We recommend setting the kernel values by executing the following commands as root:

```
# /etc/conf/bin/idtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SFNOLIM 2048
# /etc/conf/bin/idtune HFNOLIM 2048
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

We recommend tuning the system, but the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following kernel parameters can be set with **idtune**:

- SHMMAX (recommended setting: 128MB) and SHMSEG (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- SFNOLIM and HFNOLIM should be at maximum 2048.
- NPROC should be set to at least 3000/4000 (depends on number of users).
- We also recommend using the following formulas to calculate values for SEMMSL, SEMMNS, and SEMMNU:

$SEMMSL = 13$

13 is what has been found to be the best for both Progress and MySQL.

$SEMMNS = SEMMSL \times \text{number of db servers to be run on the system}$

Set SEMMNS to the value of SEMMSL multiplied by the number of database servers (maximum) that you are running on the system at one time.

$SEMMNU = SEMMNS$

Set the value of SEMMNU to equal the value of SEMMNS. You could probably set this to 75% of SEMMNS, but this is a conservative estimate.

2.13.5.10. SCO UnixWare 7.1.x and OpenUNIX 8.0.0 Notes

We recommend using the latest production release of MySQL. Should you choose to use an older release of MySQL on UnixWare 7.1.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

We have been able to compile MySQL with the following **configure** command on UnixWare 7.1.x:

```
CC="cc" CFLAGS="-I/usr/local/include" \  
CXX="CC" CXXFLAGS="-I/usr/local/include" \  
./configure --prefix=/usr/local/mysql \  
    --enable-thread-safe-client --with-berkeley-db=./bdb \  
    --with-innodb --with-openssl --with-extra-charsets=complex
```

If you want to use **gcc**, you must use **gcc 2.95.3** or newer.

```
CC=gcc CXX=g++ ... ./configure ...
```

The version of Berkeley DB that comes with either UnixWare 7.1.4 or OpenServer 6.0.0 is not used when building MySQL. MySQL instead uses its own version of Berkeley DB. The **configure** command needs to build both a static and a dynamic library in `src_directory/bdb/build_unix/`, but it does not with MySQL's own BDB version. The workaround is as follows.

1. Configure as normal for MySQL.
2. `cd bdb/build_unix/`
3. `cp -p Makefile Makefile.sav`
4. Use same options and run **../dist/configure**.
5. Run **gmake**.
6. `cp -p Makefile.sav Makefile`
7. Change to top source directory and run **gmake**.

This allows both the shared and dynamic libraries to be made and work.

SCO provides operating system patches at <ftp://ftp.sco.com/pub/unixware7> for UnixWare 7.1.1, <ftp://ftp.sco.com/pub/unixware7/713/> for UnixWare 7.1.3, <ftp://ftp.sco.com/pub/unixware7/714/> for UnixWare 7.1.4, and <ftp://ftp.sco.com/pub/openunix8> for OpenUNIX 8.0.0.

SCO provides information about security fixes at <ftp://ftp.sco.com/pub/security/OpenUNIX> for OpenUNIX and

<ftp://ftp.sco.com/pub/security/UnixWare> for UnixWare.

The UnixWare 7 file size limit is 1 TB with VXFS. Some OS utilities have a limitation of 2GB.

On UnixWare 7.1.4 you do not need to do anything to get large file support, but to enable large file support on prior versions of UnixWare 7.1.x, run **fsadm**.

```
# fsadm -Fvxfs -o largefiles /
# fsadm / * Note
# ulimit unlimited
# /etc/conf/bin/idtune SFSZLIM 0x7FFFFFFF ** Note
# /etc/conf/bin/idtune HFSZLIM 0x7FFFFFFF ** Note
# /etc/conf/bin/idbuild -B
```

* This should report "largefiles".
** 0x7FFFFFFF represents infinity for these values.

Reboot the system using shutdown.

By default, the entries in /etc/conf/cf.d/mtune are set as follows:

Value	Default	Min	Max
-----	-----	---	---
SVMMLIM	0x9000000	0x1000000	0x7FFFFFFF
HVMMLIM	0x9000000	0x1000000	0x7FFFFFFF

To make changes to the kernel, use the **idtune name parameter** command. **idtune** modifies the /etc/conf/cf.d/stune file for you. We recommend setting the kernel values by executing the following commands as root:

```
# /etc/conf/bin/idtune SDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HDATLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune HVMMLIM 0x7FFFFFFF
# /etc/conf/bin/idtune SFNOLIM 2048
# /etc/conf/bin/idtune HFNOLIM 2048
```

Then rebuild and reboot the kernel by issuing this command:

```
# /etc/conf/bin/idbuild -B && init 6
```

We recommend tuning the system, but the proper parameter values to use depend on the number of users accessing the application or database and size of the

database (that is, the used buffer pool). The following kernel parameters can be set with **idtune**:

- SHMMAX (recommended setting: 128MB) and SHMSEG (recommended setting: 15). These parameters have an influence on the MySQL database engine to create user buffer pools.
- SFNOLIM and HFNOLIM should be at maximum 2048.
- NPROC should be set to at least 3000/4000 (depends on number of users).
- We also recommend using the following formulas to calculate values for SEMMSL, SEMMNS, and SEMMNU:

SEMMSL = 13

13 is what has been found to be the best for both Progress and MySQL.

SEMMNS = SEMMSL × *number of db servers to be run on the system*

Set SEMMNS to the value of SEMMSL multiplied by the number of database servers (maximum) that you are running on the system at one time.

SEMMNU = SEMMNS

Set the value of SEMMNU to equal the value of SEMMNS. You could probably set this to 75% of SEMMNS, but this is a conservative estimate.

2.13.6. OS/2 Notes

MySQL uses quite a few open files. Because of this, you should add something like the following to your CONFIG.SYS file:

```
SET EMXOPT=-c -n -h1024
```

If you do not do this, you may encounter the following error:

```
File 'xxxx' not found (Errcode: 24)
```

When using MySQL with OS/2 Warp 3, FixPack 29 or above is required. With OS/2 Warp 4, FixPack 4 or above is required. This is a requirement of the

Pthreads library. MySQL must be installed on a partition with a type that supports long filenames, such as HPFS, FAT32, and so on.

The **INSTALL.CMD** script must be run from OS/2's own **CMD.EXE** and may not work with replacement shells such as **4OS2.EXE**.

The `scripts/mysql-install-db` script has been renamed. It is called `install.cmd` and is a REXX script, which sets up the default MySQL security settings and creates the WorkPlace Shell icons for MySQL.

Dynamic module support is compiled in but not fully tested. Dynamic modules should be compiled using the Pthreads runtime library.

```
gcc -Zdll -Zmt -Zcrtdll=pthrdrt1 -I../include -I../regex -I.. \
    -o example udf_example.cc -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

Note: Due to limitations in OS/2, UDF module name stems must not exceed eight characters. Modules are stored in the `/mysql2/udf` directory; the `safe-mysqld.cmd` script puts this directory in the `BEGINLIBPATH` environment variable. When using UDF modules, specified extensions are ignored---it is assumed to be `.udf`. For example, in Unix, the shared module might be named `example.so` and you would load a function from it like this:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example.so';
```

In OS/2, the module would be named `example.udf`, but you would not specify the module extension:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example';
```

2.14. Perl Installation Notes

Perl support for MySQL is provided by means of the DBI/DBD client interface. The interface requires Perl 5.6.1 or later. It *does not work* if you have an older version of Perl.

If you want to use transactions with Perl DBI, you need to have DBD::mysql version 1.2216 or newer. DBD::mysql 2.9003 or newer is recommended.

If you are using the MySQL 4.1 or newer client library, you must use DBD::mysql 2.9003 or newer.

Perl support is not included with MySQL distributions. You can obtain the necessary modules from <http://search.cpan.org> for Unix, or by using the ActiveState **ppm** program on Windows. The following sections describe how to do this.

Perl support for MySQL must be installed if you want to run the MySQL benchmark scripts. See [Section 7.1.4, “The MySQL Benchmark Suite”](#).

2.14.1. Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. However, if you installed MySQL from RPM files on Linux, be sure that you've installed the developer RPM. The client programs are in the client RPM, but client programming support is in the developer RPM.

If you want to install Perl support, the files you need can be obtained from the CPAN (Comprehensive Perl Archive Network) at <http://search.cpan.org>.

The easiest way to install Perl modules on Unix is to use the CPAN module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The DBD::mysql installation runs a number of tests. These tests attempt to

connect to the local MySQL server using the default username and password. (The default username is your login name on Unix, and ODBC on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

DBI requires the `Data::Dumper` module. It may be installed; if not, you should install it before installing DBI.

It is also possible to download the module distributions in the form of compressed **tar** archives and build the modules manually. For example, to unpack and build a DBI distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

```
shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
```

This command creates a directory named `DBI-VERSION`.

2. Change location into the top-level directory of the unpacked distribution:

```
shell> cd DBI-VERSION
```

3. Build the distribution and compile everything:

```
shell> perl Makefile.PL
shell> make
shell> make test
shell> make install
```

The **make test** command is important because it verifies that the module is working. Note that when you run that command during the `DBD::mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD::mysql` distribution whenever you install a new release of MySQL, particularly if you notice symptoms such as that all your DBI scripts fail after you upgrade MySQL.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: <http://servers.digitaldaze.com/extensions/perl/modules.html#modules>

Look under the heading “Installing New Modules that Require Locally Installed Modules.”

2.14.2. Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL DBD module with ActiveState Perl:

1. Get ActiveState Perl from <http://www.activestate.com/Products/ActivePerl/> and install it.
2. Open a console window (a “DOS window”).
3. If necessary, set the HTTP_proxy variable. For example, you might try a setting like this:

```
set HTTP_proxy=my.proxy.com:3128
```

4. Start the PPM program:

```
C:\> C:\perl\bin\ppm.pl
```

5. If you have not previously done so, install DBI:

```
ppm> install DBI
```

6. If this succeeds, run the following command:

```
ppm> install \  
ftp://ftp.de.uu.net/pub/CPAN/authors/id/JWIED/DBD-mysql-1.2212
```

This procedure should work with ActiveState Perl 5.6 or newer.

If you cannot get the procedure to work, you should install the MyODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;  
$dbh= DBI->connect("DBI:ODBC:$dsn", $user, $password) ||  
die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.14.3. Problems Using the Perl DBI/DBD Interface

If Perl reports that it cannot find the `../mysql/mysql.so` module, the problem is probably that Perl cannot locate the `libmysqlclient.so` shared library. You should be able to fix this problem by one of the following methods:

- Compile the `DBD::mysql` distribution with `perl Makefile.PL -static -config` rather than `perl Makefile.PL`.
- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `/usr/lib` or `/lib`).
- Modify the `-L` options used to compile `DBD::mysql` to reflect the actual location of `libmysqlclient.so`.
- On Linux, you can add the pathname of the directory where `libmysqlclient.so` is located to the `/etc/ld.so.conf` file.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable. Some systems use `LD_LIBRARY_PATH` instead.

Note that you may also need to modify the `-L` options if there are other libraries that the linker fails to find. For example, if the linker cannot find `libc` because it is in `/lib` and the link command specifies `-L/usr/lib`, change the `-L` option to `-L/lib` or add `-L/lib` to the existing link command.

If you get the following errors from `DBD::mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'  
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the `mysql.so` library gets built (check the output from `make` for `mysql.so` when you compile the Perl client). The `-L` option should specify the pathname of the directory where `libgcc.a` is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

You may see the following error from `DBD::mysql` when you run the tests:

```
t/00base.....install_driver(mysql) failed:
Can't load '../blib/arch/auto/DBD/mysql/mysql.so' for module DBD::my
../blib/arch/auto/DBD/mysql/mysql.so: undefined symbol:
uncompress at /usr/lib/perl5/5.00503/i586-linux/DynaLoader.pm line 1
```

This means that you need to include the `-lz` compression library on the link line. That can be done by changing the following line in the file `lib/DBD/mysql/Install.pm`:

```
$sysliblist .= " -lm";
```

Change that line to:

```
$sysliblist .= " -lm -lz";
```

After this, you *must* run **make realclean** and then proceed with the installation from the beginning.

If you want to install DBI on SCO, you have to edit the `Makefile` in `DBI-xxx` and each subdirectory. Note that the following assumes **gcc** 2.95.2 or newer:

OLD:	NEW:
CC = cc	CC = gcc
CCCDLFLAGS = -KPIC -w1, -Bexport	CCCDLFLAGS = -fpic
CCDLFLAGS = -w1, -Bexport	CCDLFLAGS =
LD = ld	LD = gcc -G -fpic
LDDLFLAGS = -G -L/usr/local/lib	LDDLFLAGS = -L/usr/local/lib
LDFLAGS = -belf -L/usr/local/lib	LDFLAGS = -L/usr/local/lib
LD = ld	LD = gcc -G -fpic
OPTIMISE = -Od	OPTIMISE = -O1
OLD:	
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SC05 -I/usr/local/include	
NEW:	
CCCFLAGS = -U M_XENIX -DPERL_SC05 -I/usr/local/include	

These changes are necessary because the Perl dynaloader does not load the DBI modules if they were compiled with **icc** or **cc**.

If you want to use the Perl module on a system that does not support dynamic linking (such as SCO), you can generate a static version of Perl that includes DBI

and `DBD::mysql`. The way this works is that you generate a version of Perl with the DBI code linked in and install it on top of your current Perl. Then you use that to build a version of Perl that additionally has the DBD code linked in, and install that.

On SCO, you must have the following environment variables set:

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
```

Or:

```
LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:\
    /usr/skunk/man:
```

First, create a Perl that includes a statically linked DBI module by running these commands in the directory where your DBI distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Then you must install the new Perl. The output of **make perl** indicates the exact **make** command you need to execute to perform the installation. On SCO, this is **make -f Makefile.aperl inst_perl MAP_TARGET=perl**.

Next, use the just-created Perl to create another Perl that also includes a statically linked `DBD::mysql` by running these commands in the directory where your `DBD::mysql` distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Finally, you should install this new Perl. Again, the output of **make perl** indicates the command to use.

Chapter 3. Tutorial

Table of Contents

- [3.1. Connecting to and Disconnecting from the Server](#)
- [3.2. Entering Queries](#)
- [3.3. Creating and Using a Database](#)
 - [3.3.1. Creating and Selecting a Database](#)
 - [3.3.2. Creating a Table](#)
 - [3.3.3. Loading Data into a Table](#)
 - [3.3.4. Retrieving Information from a Table](#)
- [3.4. Getting Information About Databases and Tables](#)
- [3.5. Using **mysql** in Batch Mode](#)
- [3.6. Examples of Common Queries](#)
 - [3.6.1. The Maximum Value for a Column](#)
 - [3.6.2. The Row Holding the Maximum of a Certain Column](#)
 - [3.6.3. Maximum of Column per Group](#)
 - [3.6.4. The Rows Holding the Group-wise Maximum of a Certain Field](#)
 - [3.6.5. Using User-Defined Variables](#)
 - [3.6.6. Using Foreign Keys](#)
 - [3.6.7. Searching on Two Keys](#)
 - [3.6.8. Calculating Visits Per Day](#)
 - [3.6.9. Using `AUTO INCREMENT`](#)
- [3.7. Queries from the Twin Project](#)
 - [3.7.1. Find All Non-distributed Twins](#)
 - [3.7.2. Show a Table of Twin Pair Status](#)
- [3.8. Using MySQL with Apache](#)

This chapter provides a tutorial introduction to MySQL by showing how to use the **mysql** client program to create and use a simple database. **mysql** (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. **mysql** may also be used in batch mode: you place your queries in a file beforehand, then tell **mysql** to execute the contents of the file. Both ways of using **mysql** are covered here.

To see a list of options provided by **mysql**, invoke it with the `--help` option:

```
shell> mysql --help
```

This chapter assumes that **mysql** is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as [Chapter 5, Database Administration](#).)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1. Connecting to and Disconnecting from the Server

To connect to the server, you will usually need to provide a MySQL user name when you invoke **mysql** and, most likely, a password. If the server runs on a machine other than the one where you log in, you will also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: *****
```

host and user represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The ***** represents your password; enter it when **mysql** displays the Enter password: prompt.

If that works, you should see some introductory information followed by a mysql> prompt:

```
shell> mysql -h host -u user -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.0.25-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The mysql> prompt tells you that **mysql** is ready for you to enter commands.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell< mysql -u user -p
```

If, when you attempt to log in, you get an error message such as ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2), it means that that MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of [Chapter 2, Installing and Upgrading MySQL](#) that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see [Section A.2, “Common Errors When Using MySQL Programs”](#).

Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking **mysql** without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing **QUIT** (or **\q**) at the `mysql>` prompt:

```
mysql> QUIT  
Bye
```

On Unix, you can also disconnect by pressing Control-D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the `mysql>` prompt.

3.2. Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that's okay. At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how **mysql** works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----+-----+
| VERSION()          | CURRENT_DATE |
+-----+-----+
| 5.0.7-beta-Max    | 2005-07-11  |
+-----+-----+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about **mysql**:

- A command normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a command, **mysql** sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another command.
- **mysql** displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), **mysql** labels the column using the expression itself.

- **mysql** shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the “rows in set” line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here's another query. It demonstrates that you can use **mysql** as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
| 0.70710678118655 | 25 |
+-----+-----+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+-----+
| VERSION() |
+-----+
| 5.0.7-beta-Max |
+-----+
1 row in set (0.00 sec)

+-----+
| NOW() |
+-----+
| 2005-07-11 17:59:36 |
+-----+
1 row in set (0.00 sec)
```

A command need not be given all on a single line, so lengthy commands that

require several lines are not a problem. **mysql** determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, **mysql** accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here's a simple multiple-line statement:

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
+-----+-----+
| USER()          | CURRENT_DATE |
+-----+-----+
| jon@localhost  | 2005-07-11   |
+-----+-----+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how **mysql** indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what **mysql** is waiting for.

If you decide you do not want to execute a command that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
-> USER()
-> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that **mysql** is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that **mysql** is in:

Prompt	Meaning
<code>mysql></code>	Ready for new command.
<code>-></code>	Waiting for next line of multiple-line command.
<code>'></code>	Waiting for next line, waiting for completion of a string that began with a single quote ('').

">	Waiting for next line, waiting for completion of a string that began with a double quote ('').
`>	Waiting for next line, waiting for completion of an identifier that began with a backtick (` `).
/*>	Waiting for next line, waiting for completion of a comment that began with /*.

In the MySQL 5.0 series, the /*> prompt was implemented in MySQL 5.0.6.

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, **mysql** waits for more input:

```
mysql> SELECT USER()  
->
```

If this happens to you (you think you've entered a statement but the only response is a -> prompt), most likely **mysql** is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and **mysql** executes it:

```
mysql> SELECT USER()  
-> ;  
+-----+  
| USER() |  
+-----+  
| jon@localhost |  
+-----+
```

The '>' and '>' prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either '' or "" characters (for example, 'hello' or "goodbye"), and **mysql** lets you enter strings that span multiple lines. When you see a '>' or '>' prompt, it means that you have entered a line containing a string that begins with a '' or "" quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
```

'>

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that **mysql** expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quote mark.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type `\c` in this case, because **mysql** interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so **mysql** knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;  
      '> '\c  
mysql>
```

The prompt changes back to `mysql>`, indicating that **mysql** is ready for a new command.

The ``>` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and ``>` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by **mysql** — including a line containing `QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current command.

3.3. Creating and Using a Database

Once you know how to enter commands, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL Web site. It is available in both compressed **tar** file and Zip formats at <http://dev.mysql.com/doc/>.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;  
+-----+  
| Database |  
+-----+  
| mysql   |  
| test    |  
| tmp     |  
+-----+
```

The list of databases is probably different on your machine, but the `mysql` and

test databases are likely to be among them. The `mysql` database is required because it describes user access privileges. The `test` database is often provided as a workspace for users to try things out.

Note that you may not see all databases if you do not have the `SHOW DATABASES` privilege. See [Section 13.5.1.3, “GRANT Syntax”](#).

If the `test` database exists, try to access it:

```
mysql> USE test  
Database changed
```

Note that `USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours `menagerie`. The administrator needs to execute a command like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host'
```

where `your_mysql_name` is the MySQL user name assigned to you and `your_client_host` is the host from which you connect to the server.

3.3.1. Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case sensitive (unlike SQL keywords), so you must always refer to your database as `menagerie`, not as `Menagerie`, `MENAGERIE`, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, our recommended best practice is always to use the same lettercase that was

used when the database was created.)

Note: If you get an error such as ERROR 1044 (42000): Access denied for user 'monty'@'localhost' to database 'menagerie' when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see [Section 5.8, “The MySQL Access Privilege System”](#).

Creating a database does not select it for use; you must do that explicitly. To make `menagerie` the current database, use this command:

```
mysql> USE menagerie;  
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a **mysql** session. You can do this by issuing a `USE` statement as shown in the example. Alternatively, you can select the database on the command line when you invoke **mysql**. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie  
Enter password: *****
```

Note that `menagerie` in the command just shown is **not** your password. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (for example, as `-pmypassword`, *not* as `-p mypassword`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

3.3.2. Creating a Table

Creating the database is the easy part, but at this point it's empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;  
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the pet table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it's not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it's better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the pet table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),  
-> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the name, owner, and species columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be 20. You can normally pick any length from 1 to 65535, whatever seems most reasonable to you. (**Note:** Prior to MySQL 5.0.3, the upper limit was 255.) If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.

Several types of values can be chosen to represent sex in animal records, such as 'm' and 'f', or perhaps 'male' and 'female'. It is simplest to use the single characters 'm' and 'f'.

The use of the DATE data type for the birth and death columns is a fairly obvious choice.

Once you have created a table, SHOW TABLES should produce some output:

```
mysql> SHOW TABLES;
+-----+
| Tables in menagerie |
+-----+
| pet                  |
+-----+
```

To verify that your table was created the way you expected, use a DESCRIBE statement:

```
mysql> DESCRIBE pet;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | varchar(20)   | YES  |     | NULL    |       |
| owner | varchar(20)   | YES  |     | NULL    |       |
| species | varchar(20)  | YES  |     | NULL    |       |
| sex   | char(1)       | YES  |     | NULL    |       |
| birth | date          | YES  |     | NULL    |       |
| death | date          | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

You can use DESCRIBE any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see [Chapter 11, Data Types](#).

3.3.3. Loading Data into a Table

After creating your table, you need to populate it. The LOAD DATA and INSERT statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in 'YYYY-MM-DD' format; this may be different from what

you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler      Gwen      bird      \N      1997-12-09      \N
```

To load the text file `pet.txt` into the `pet` table, use this command:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

Note that if you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet  
      -> LINES TERMINATED BY '\r\n';
```

(On an Apple machine running OS X, you would likely want to use `LINES TERMINATED BY '\r'`.)

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt` properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See [Section 5.7.4, “Security Issues with `LOAD DATA LOCAL`”](#), for information on how to change this.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose that Diane gets a new hamster named “Puffball.” You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
-> VALUES ('Puffball', 'Diane', 'hamster', 'f', '1999-03-30', NULL);
```

Note that string and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

3.3.4. Retrieving Information from a Table

The `SELECT` statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

what_to_select indicates what you want to see. This can be a list of columns, or `*` to indicate “all columns.” *which_table* indicates the table from which you want to retrieve data. The `WHERE` clause is optional. If it is present, *conditions_to_satisfy* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1. Selecting All Data

The simplest form of SELECT retrieves everything from a table:

```
mysql> SELECT * FROM pet;
+-----+-----+-----+-----+-----+-----+
| name      | owner  | species | sex  | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Fluffy    | Harold | cat      | f    | 1993-02-04 | NULL       |
| Claws     | Gwen   | cat      | m    | 1994-03-17 | NULL       |
| Buffy     | Harold | dog      | f    | 1989-05-13 | NULL       |
| Fang      | Benny  | dog      | m    | 1990-08-27 | NULL       |
| Bowser    | Diane  | dog      | m    | 1979-08-31 | 1995-07-29 |
| Chirpy    | Gwen   | bird     | f    | 1998-09-11 | NULL       |
| Whistler  | Gwen   | bird     | NULL | 1997-12-09 | NULL       |
| Slim      | Benny  | snake    | m    | 1996-04-29 | NULL       |
| Puffball  | Diane  | hamster  | f    | 1999-03-30 | NULL       |
+-----+-----+-----+-----+-----+-----+
```

This form of SELECT is useful if you want to review your entire table, for example, after you've just loaded it with your initial dataset. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file `pet.txt` to correct the error, then empty the table and reload it using `DELETE` and `LOAD DATA`:

```
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
```

However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an `UPDATE` statement:

```
mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser'
```

The `UPDATE` changes only the record in question and does not require you to reload the table.

3.3.4.2. Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the `WHERE` clause from the `SELECT` statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+-----+-----+-----+-----+-----+-----+
| name   | owner | species | sex  | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Bowser | Diane | dog      | m    | 1989-08-31 | 1995-07-29 |
+-----+-----+-----+-----+-----+-----+
```

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as 'bowser', 'BOWSER', and so forth. The query result is the same.

You can specify conditions on any column, not just name. For example, if you want to know which animals were born during or after 1998, test the birth column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+-----+-----+-----+-----+-----+-----+
| name      | owner | species | sex  | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Chirpy    | Gwen  | bird     | f    | 1998-09-11 | NULL       |
| Puffball  | Diane | hamster  | f    | 1999-03-30 | NULL       |
+-----+-----+-----+-----+-----+-----+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex  | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Buffy  | Harold | dog      | f    | 1989-05-13 | NULL       |
+-----+-----+-----+-----+-----+-----+
```

The preceding query uses the AND logical operator. There is also an OR operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird'
+-----+-----+-----+-----+-----+-----+
| name      | owner  | species | sex   | birth      | death  |
+-----+-----+-----+-----+-----+-----+
| Chirpy    | Gwen   | bird    | f     | 1998-09-11 | NULL   |
| Whistler  | Gwen   | bird    | NULL  | 1997-12-09 | NULL   |
| Slim      | Benny  | snake   | m     | 1996-04-29 | NULL   |
+-----+-----+-----+-----+-----+-----+
```

AND and OR may be intermixed, although AND has higher precedence than OR. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
-> OR (species = 'dog' AND sex = 'f');
+-----+-----+-----+-----+-----+-----+
| name  | owner  | species | sex   | birth      | death  |
+-----+-----+-----+-----+-----+-----+
| Claws | Gwen   | cat     | m     | 1994-03-17 | NULL   |
| Buffy | Harold | dog     | f     | 1989-05-13 | NULL   |
+-----+-----+-----+-----+-----+-----+
```

3.3.4.3. Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the name and birth columns:

```
mysql> SELECT name, birth FROM pet;
+-----+-----+
| name      | birth      |
+-----+-----+
| Fluffy    | 1993-02-04 |
| Claws     | 1994-03-17 |
| Buffy     | 1989-05-13 |
| Fang      | 1990-08-27 |
| Bowser    | 1989-08-31 |
| Chirpy    | 1998-09-11 |
| Whistler  | 1997-12-09 |
| Slim      | 1996-04-29 |
| Puffball  | 1999-03-30 |
+-----+-----+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
```

```
+-----+
| owner |
+-----+
| Harold |
| Gwen  |
| Harold |
| Benny  |
| Diane  |
| Gwen  |
| Gwen  |
| Benny  |
| Diane  |
+-----+
```

Notice that the query simply retrieves the owner column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
```

```
+-----+
| owner |
+-----+
| Benny  |
| Diane  |
| Gwen  |
| Harold |
+-----+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
       -> WHERE species = 'dog' OR species = 'cat';
```

```
+-----+-----+-----+
| name   | species | birth   |
+-----+-----+-----+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+-----+-----+-----+
```

3.3.4.4. Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It's often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+-----+-----+
| name      | birth      |
+-----+-----+
| Buffy     | 1989-05-13 |
| Bowser    | 1989-08-31 |
| Fang      | 1990-08-27 |
| Fluffy    | 1993-02-04 |
| Claws     | 1994-03-17 |
| Slim      | 1996-04-29 |
| Whistler  | 1997-12-09 |
| Chirpy    | 1998-09-11 |
| Puffball  | 1999-03-30 |
+-----+-----+
```

On character type columns, sorting — like all other comparison operations — is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY` like so: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+-----+-----+
| name      | birth      |
+-----+-----+
| Puffball  | 1999-03-30 |
| Chirpy    | 1998-09-11 |
| Whistler  | 1997-12-09 |
| Slim      | 1996-04-29 |
| Claws     | 1994-03-17 |
| Fluffy    | 1993-02-04 |
| Fang      | 1990-08-27 |
| Bowser    | 1989-08-31 |
| Buffy     | 1989-05-13 |
+-----+-----+
```

```
+-----+-----+
```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet  
-> ORDER BY species, birth DESC;
```

```
+-----+-----+-----+  
| name      | species | birth      |  
+-----+-----+-----+  
| Chirpy    | bird    | 1998-09-11 |  
| Whistler  | bird    | 1997-12-09 |  
| Claws     | cat     | 1994-03-17 |  
| Fluffy    | cat     | 1993-02-04 |  
| Fang      | dog     | 1990-08-27 |  
| Bowser    | dog     | 1989-08-31 |  
| Buffy     | dog     | 1989-05-13 |  
| Puffball  | hamster | 1999-03-30 |  
| Slim      | snake   | 1996-04-29 |  
+-----+-----+-----+
```

Note that the DESC keyword applies only to the column name immediately preceding it (birth); it does not affect the species column sort order.

3.3.4.5. Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, compute the difference in the year part of the current date and the birth date, then subtract one if the current date occurs earlier in the calendar year than the birth date. The following query shows, for each pet, the birth date, the current date, and the age in years.

```
mysql> SELECT name, birth, CURDATE(),  
-> (YEAR(CURDATE())-YEAR(birth))  
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))  
-> AS age  
-> FROM pet;
```

```
+-----+-----+-----+-----+  
| name      | birth      | CURDATE() | age |  
+-----+-----+-----+-----+
```

Fluffy	1993-02-04	2003-08-19	10
Claws	1994-03-17	2003-08-19	9
Buffy	1989-05-13	2003-08-19	14
Fang	1990-08-27	2003-08-19	12
Bowser	1989-08-31	2003-08-19	13
Chirpy	1998-09-11	2003-08-19	4
Whistler	1997-12-09	2003-08-19	5
Slim	1996-04-29	2003-08-19	7
Puffball	1999-03-30	2003-08-19	4

Here, YEAR() pulls out the year part of a date and RIGHT() pulls off the rightmost five characters that represent the MM-DD (calendar year) part of the date. The part of the expression that compares the MM-DD values evaluates to 1 or 0, which adjusts the year difference down a year if CURDATE() occurs earlier in the year than birth. The full expression is somewhat ungainly, so an *alias* (age) is used to make the output column label more meaningful.

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an ORDER BY name clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
-> AS age
-> FROM pet ORDER BY name;
+-----+-----+-----+-----+
| name      | birth      | CURDATE() | age |
+-----+-----+-----+-----+
| Bowser    | 1989-08-31 | 2003-08-19 | 13 |
| Buffy     | 1989-05-13 | 2003-08-19 | 14 |
| Chirpy    | 1998-09-11 | 2003-08-19 | 4  |
| Claws     | 1994-03-17 | 2003-08-19 | 9  |
| Fang      | 1990-08-27 | 2003-08-19 | 12 |
| Fluffy    | 1993-02-04 | 2003-08-19 | 10 |
| Puffball  | 1999-03-30 | 2003-08-19 | 4  |
| Slim      | 1996-04-29 | 2003-08-19 | 7  |
| Whistler  | 1997-12-09 | 2003-08-19 | 5  |
+-----+-----+-----+-----+
```

To sort the output by age rather than name, just use a different ORDER BY clause:

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
```

```

-> AS age
-> FROM pet ORDER BY age;
+-----+-----+-----+-----+
| name      | birth      | CURDATE()  | age  |
+-----+-----+-----+-----+
| Chirpy    | 1998-09-11 | 2003-08-19 | 4    |
| Puffball  | 1999-03-30 | 2003-08-19 | 4    |
| Whistler  | 1997-12-09 | 2003-08-19 | 5    |
| Slim      | 1996-04-29 | 2003-08-19 | 7    |
| Claws     | 1994-03-17 | 2003-08-19 | 9    |
| Fluffy    | 1993-02-04 | 2003-08-19 | 10   |
| Fang      | 1990-08-27 | 2003-08-19 | 12   |
| Bowser    | 1989-08-31 | 2003-08-19 | 13   |
| Buffy     | 1989-05-13 | 2003-08-19 | 14   |
+-----+-----+-----+-----+

```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the death value is NULL. Then, for those with non-NULL values, compute the difference between the death and birth values:

```

mysql> SELECT name, birth, death,
-> (YEAR(death)-YEAR(birth)) - (RIGHT(death,5)<RIGHT(birth,5))
-> AS age
-> FROM pet WHERE death IS NOT NULL ORDER BY age;
+-----+-----+-----+-----+
| name      | birth      | death      | age  |
+-----+-----+-----+-----+
| Bowser    | 1989-08-31 | 1995-07-29 | 5    |
+-----+-----+-----+-----+

```

The query uses `death IS NOT NULL` rather than `death <> NULL` because NULL is a special value that cannot be compared using the usual comparison operators. This is discussed later. See [Section 3.3.4.6, “Working with NULL Values”](#).

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the birth column. MySQL provides several functions for extracting parts of dates, such as `YEAR()`, `MONTH()`, and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both birth and `MONTH(birth)`:

```

mysql> SELECT name, birth, MONTH(birth) FROM pet;
+-----+-----+-----+
| name      | birth      | MONTH(birth) |
+-----+-----+-----+

```

Fluffy	1993-02-04	2
Claws	1994-03-17	3
Buffy	1989-05-13	5
Fang	1990-08-27	8
Bowser	1989-08-31	8
Chirpy	1998-09-11	9
Whistler	1997-12-09	12
Slim	1996-04-29	4
Puffball	1999-03-30	3

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is 4 and you can look for animals born in May (month 5) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

name	birth
Buffy	1989-05-13

There is a small complication if the current month is December. You cannot merely add one to the month number (12) and look for animals born in month 13, because there is no such month. Instead, you look for animals born in January (month 1).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` allows you to add a time interval to a given date. If you add a month to the value of `CURDATE()`, then extract the month part with `MONTH()`, the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
-> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(), INTERVAL 1 MONT
```

A different way to accomplish the same task is to add 1 to get the next month after the current one after using the modulo function (`MOD`) to wrap the month value to 0 if it is currently 12:

```
mysql> SELECT name, birth FROM pet
-> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

Note that MONTH returns a number between 1 and 12. And MOD(something, 12) returns a number between 0 and 11. So the addition has to be after the MOD(), otherwise we would go from November (11) to January (1).

3.3.4.6. Working with NULL Values

The NULL value can be surprising until you get used to it. Conceptually, NULL means “a missing unknown value” and it is treated somewhat differently from other values. To test for NULL, you cannot use the arithmetic comparison operators such as =, <, or <>. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+-----+-----+-----+-----+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+-----+-----+-----+-----+
|      NULL |      NULL |      NULL |      NULL |
+-----+-----+-----+-----+
```

Clearly you get no meaningful results from these comparisons. Use the IS NULL and IS NOT NULL operators instead:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----+-----+
| 1 IS NULL | 1 IS NOT NULL |
+-----+-----+
|          0 |          1 |
+-----+-----+
```

Note that in MySQL, 0 or NULL means false and anything else means true. The default truth value from a boolean operation is 1.

This special treatment of NULL is why, in the previous section, it was necessary to determine which animals are no longer alive using death IS NOT NULL instead of death <> NULL.

Two NULL values are regarded as equal in a GROUP BY.

When doing an ORDER BY, NULL values are presented first if you do ORDER BY ... ASC and last if you do ORDER BY ... DESC.

A common error when working with NULL is to assume that it is not possible to

insert a zero or an empty string into a column defined as NOT NULL, but this is not the case. These are in fact values, whereas NULL means “not having a value.” You can test this easily enough by using IS [NOT] NULL as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----+-----+-----+-----+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----+-----+-----+-----+
|          0 |             1 |           0 |             1 |
+-----+-----+-----+-----+
```

Thus it is entirely possible to insert a zero or empty string into a NOT NULL column, as these are in fact NOT NULL. See [Section A.5.3, “Problems with NULL Values”](#).

3.3.4.7. Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as **vi**, **grep**, and **sed**.

SQL pattern matching allows you to use ‘_’ to match any single character and ‘%’ to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. Note that you do not use = or <> when you use SQL patterns; use the LIKE or NOT LIKE comparison operators instead.

To find names beginning with ‘b’:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex  | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+-----+-----+-----+-----+-----+-----+
```

To find names ending with ‘fy’:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex  | birth      | death      |
+-----+-----+-----+-----+-----+-----+
```

Fluffy	Harold	cat	f	1993-02-04	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

To find names containing a 'w':

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29
Whistler	Gwen	bird	NULL	1997-12-09	NULL

To find names containing exactly five characters, use five instances of the '_' pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the REGEXP and NOT REGEXP operators (or RLIKE and NOT RLIKE, which are synonyms).

Some characteristics of extended regular expressions are:

- '.' matches any single character.
- A character class '[...]' matches any character within the brackets. For example, '[abc]' matches 'a', 'b', or 'c'. To name a range of characters, use a dash. '[a-z]' matches any letter, whereas '[0-9]' matches any digit.
- '*' matches zero or more instances of the thing preceding it. For example, 'x*' matches any number of 'x' characters, '[0-9]*' matches any number of digits, and '.*' matches any number of anything.
- A REGEXP pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a LIKE pattern match, which succeeds

only if the pattern matches the entire value.)

- To anchor a pattern so that it must match the beginning or end of the value being tested, use ‘^’ at the beginning or ‘\$’ at the end of the pattern.

To demonstrate how extended regular expressions work, the LIKE queries shown previously are rewritten here to use REGEXP.

To find names beginning with ‘b’, use ‘^’ to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
```

name	owner	species	sex	birth	death
Buffy	Harold	dog	f	1989-05-13	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29

If you really want to force a REGEXP comparison to be case sensitive, use the BINARY keyword to make one of the strings a binary string. This query matches only lowercase ‘b’ at the beginning of a name:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

To find names ending with ‘fy’, use ‘\$’ to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
```

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	NULL
Buffy	Harold	dog	f	1989-05-13	NULL

To find names containing a ‘w’, use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
```

name	owner	species	sex	birth	death
Claws	Gwen	cat	m	1994-03-17	NULL
Bowser	Diane	dog	m	1989-08-31	1995-07-29
Whistler	Gwen	bird	NULL	1997-12-09	NULL

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

To find names containing exactly five characters, use ‘^’ and ‘\$’ to match the beginning and end of the name, and five instances of ‘.’ in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex   | birth      | death  |
+-----+-----+-----+-----+-----+-----+
| Claws  | Gwen  | cat     | m    | 1994-03-17 | NULL   |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL   |
+-----+-----+-----+-----+-----+-----+
```

You could also write the previous query using the {n} (“repeat-*n*-times”) operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
+-----+-----+-----+-----+-----+-----+
| name   | owner  | species | sex   | birth      | death  |
+-----+-----+-----+-----+-----+-----+
| Claws  | Gwen  | cat     | m    | 1994-03-17 | NULL   |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL   |
+-----+-----+-----+-----+-----+-----+
```

[Appendix G, Regular Expressions](#), provides more information about the syntax for regular expressions.

3.3.4.8. Counting Rows

Databases are often used to answer the question, “How often does a certain type of data occur in a table?” For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as “How many rows are in the pet table?” because there is one record per pet. `COUNT(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
+-----+
```

```
| COUNT(*) |
+-----+
|          9 |
+-----+
```

Earlier, you retrieved the names of the people who owned pets. You can use COUNT() if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+-----+-----+
| owner  | COUNT(*) |
+-----+-----+
| Benny  |         2 |
| Diane  |         2 |
| Gwen   |         3 |
| Harold |         2 |
+-----+-----+
```

Note the use of GROUP BY to group all records for each owner. Without it, all you get is an error message:

```
mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...
with no GROUP columns is illegal if there is no GROUP BY clause
```

COUNT() and GROUP BY are useful for characterizing your data in various ways. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+-----+-----+
| species | COUNT(*) |
+-----+-----+
| bird    |         2 |
| cat     |         2 |
| dog     |         3 |
| hamster |         1 |
| snake   |         1 |
+-----+-----+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+-----+-----+
```

sex	COUNT(*)
NULL	1
f	4
m	4

(In this output, NULL indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
```

species	sex	COUNT(*)
bird	NULL	1
bird	f	1
cat	f	1
cat	m	1
dog	f	1
dog	m	2
hamster	f	1
snake	m	1

You need not retrieve an entire table when you use COUNT(). For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
-> WHERE species = 'dog' OR species = 'cat'
-> GROUP BY species, sex;
```

species	sex	COUNT(*)
cat	f	1
cat	m	1
dog	f	1
dog	m	2

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
-> WHERE sex IS NOT NULL
-> GROUP BY species, sex;
```

species	sex	COUNT(*)
bird	f	1
cat	f	1
cat	m	1
dog	f	1
dog	m	2
hamster	f	1
snake	m	1

3.3.4.9. Using More Than one Table

The pet table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs:

- To contain the pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the CREATE TABLE statement for the event table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
-> type VARCHAR(15), remark VARCHAR(255));
```

As with the pet table, it's easiest to load the initial records by creating a tab-delimited text file containing the information:

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened

Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the pet table, you should be able to perform retrievals on the records in the event table; the principles are the same. But when is the event table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the event table, but to calculate her age on that date you need her birth date, which is stored in the pet table. This means the query requires both tables:

```
mysql> SELECT pet.name,
-> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS
-> remark
-> FROM pet INNER JOIN event
-> ON pet.name = event.name
-> WHERE event.type = 'litter';
```

```
+-----+-----+-----+
| name   | age  | remark                               |
+-----+-----+-----+
| Fluffy |    2 | 4 kittens, 3 female, 1 male         |
| Buffy  |    4 | 5 puppies, 2 female, 3 male         |
| Buffy  |    5 | 3 puppies, 3 female                 |
+-----+-----+-----+
```

There are several things to note about this query:

- The FROM clause joins two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to

specify how records in one table can be matched to records in the other. This is easy because they both have a name column. The query uses WHERE clause to match up records in the two tables based on the name values.

The query uses an INNER JOIN to combine the tables. An INNER JOIN allows for rows from either table to appear in the result if and only if both tables meet the conditions specified in the ON clause. In this example, the ON clause specifies that the name column in the pet table must match the name column in the event table. If a name appears in one table but not the other, the row will not appear in the result because the condition in the ON clause fails.

- Because the name column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the pet table with itself to produce candidate pairs of males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
-> FROM pet AS p1 INNER JOIN pet AS p2
-> ON p1.species = p2.species AND p1.sex = 'f' AND p2.sex = 'm'
+-----+-----+-----+-----+-----+
| name   | sex   | name   | sex   | species |
+-----+-----+-----+-----+-----+
| Fluffy | f     | Claws  | m     | cat     |
| Buffy  | f     | Fang   | m     | dog     |
| Buffy  | f     | Bowser | m     | dog     |
+-----+-----+-----+-----+-----+
```

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4. Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` function:

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| menagerie |
+-----+
```

If you have not yet selected any database, the result is `NULL`.

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this command:

```
mysql> SHOW TABLES;
+-----+
| Tables in menagerie |
+-----+
| event                |
| pet                  |
+-----+
```

If you want to find out about the structure of a table, the `DESCRIBE` command is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | varchar(20)   | YES  |     | NULL    |       |
| owner | varchar(20)   | YES  |     | NULL    |       |
| species | varchar(20)  | YES  |     | NULL    |       |
| sex   | char(1)       | YES  |     | NULL    |       |
| birth | date          | YES  |     | NULL    |       |
```

```
| death | date | YES | | NULL | | |
+-----+-----+-----+-----+-----+-----+
```

Field indicates the column name, Type is the data type for the column, NULL indicates whether the column can contain NULL values, Key indicates whether the column is indexed, and Default specifies the column's default value.

If you have indexes on a table, `SHOW INDEX FROM tbl_name` produces information about them.

3.5. Using `mysql` in Batch Mode

In the previous sections, you used `mysql` interactively to enter queries and view the results. You can also run `mysql` in batch mode. To do this, put the commands you want to run in a file, then tell `mysql` to read its input from the file:

```
shell> mysql < batch-file
```

If you are running `mysql` under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file  
Enter password: *****
```

When you use `mysql` this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force` command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script allows you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line commands or multiple-statement sequences of commands. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

```
shell> mysql < batch-file | more
```

- You can catch the output in a file for further processing:

```
shell> mysql < batch-file > mysql.out
```

- You can distribute your script to other people so that they can also run the commands.
- Some situations do not allow for interactive use, for example, when you run a query from a **cron** job. In this case, you must use batch mode.

The default output format is different (more concise) when you run **mysql** in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when **mysql** is run interactively:

```
+-----+
| species |
+-----+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+-----+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the commands that are executed, use `mysql -vvv`.

You can also use scripts from the **mysql** prompt by using the `source` or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

3.6. Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then `(article, dealer)` is a primary key for the records.

Start the command-line tool **mysql** and select a database:

```
shell> mysql your-database-name
```

(In most MySQL installations, you can use the database named `test`).

You can create and populate the example table with these statements:

```
mysql> CREATE TABLE shop (  
-> article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,  
-> dealer CHAR(20) DEFAULT '' NOT NULL,  
-> price DOUBLE(16,2) DEFAULT '0.00' NOT NULL,  
-> PRIMARY KEY(article, dealer));  
mysql> INSERT INTO shop VALUES  
-> (1, 'A', 3.45), (1, 'B', 3.99), (2, 'A', 10.99), (3, 'B', 1.45),  
-> (3, 'C', 1.69), (3, 'D', 1.25), (4, 'D', 19.95);
```

After issuing the statements, the table should have the following contents:

```
mysql> SELECT * FROM shop;  
+-----+-----+-----+  
| article | dealer | price |  
+-----+-----+-----+  
| 0001 | A | 3.45 |  
| 0001 | B | 3.99 |  
| 0002 | A | 10.99 |  
| 0003 | B | 1.45 |  
| 0003 | C | 1.69 |  
| 0003 | D | 1.25 |  
| 0004 | D | 19.95 |  
+-----+-----+-----+
```

3.6.1. The Maximum Value for a Column

“What's the highest item number?”

```
SELECT MAX(article) AS article FROM shop;
```

```
+-----+
| article |
+-----+
|      4 |
+-----+
```

3.6.2. The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);
```

Another solution is to sort all rows descending by price and get only the first row using the MySQL-specific LIMIT clause:

```
SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Note: If there were several most expensive articles, each with a price of 19.95, the LIMIT solution would show only one of them.

3.6.3. Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article
```

```
+-----+-----+
| article | price |
+-----+-----+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+-----+-----+
```

3.6.4. The Rows Holding the Group-wise Maximum of a Certain Field

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article);
```

3.6.5. Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See [Section 9.3, “User-Defined Variables”](#).)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM sho
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price
+-----+-----+-----+
| article | dealer | price |
+-----+-----+-----+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+-----+-----+-----+
```

3.6.6. Using Foreign Keys

In MySQL, InnoDB tables support checking of foreign key constraints. See [Section 14.2, “The InnoDB Storage Engine”](#), and [Section 1.9.5.5, “Foreign Keys”](#).

A foreign key constraint is not required merely to join two tables. For storage engines other than InnoDB, it is possible when defining a column to use a REFERENCES tbl_name(col_name) clause, which has no actual effect, and serves only as a memo or comment to you that the column which you are currently defining is intended to refer to a column in another table. It is extremely

important to realize when using this syntax that:

- MySQL does not perform any sort of CHECK to make sure that *col_name* actually exists in *tbl_name* (or even that *tbl_name* itself exists).
- MySQL does not perform any sort of action on *tbl_name* such as deleting rows in response to actions taken on rows in the table which you are defining; in other words, this syntax induces no ON DELETE or ON UPDATE behavior whatsoever. (Although you can write an ON DELETE or ON UPDATE clause as part of the REFERENCES clause, it is also ignored.)
- This syntax creates a *column*; it does **not** create any sort of index or key.
- This syntax will cause an error if used in trying to define an InnoDB table.

You can use a column so created as a join column, as shown here:

```
CREATE TABLE person (  
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    name CHAR(60) NOT NULL,  
    PRIMARY KEY (id)  
);  
  
CREATE TABLE shirt (  
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,  
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,  
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),  
    PRIMARY KEY (id)  
);  
  
INSERT INTO person VALUES (NULL, 'Antonio Paz');  
  
SELECT @last := LAST_INSERT_ID();  
  
INSERT INTO shirt VALUES  
(NULL, 'polo', 'blue', @last),  
(NULL, 'dress', 'white', @last),  
(NULL, 't-shirt', 'blue', @last);  
  
INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');  
  
SELECT @last := LAST_INSERT_ID();  
  
INSERT INTO shirt VALUES  
(NULL, 'dress', 'orange', @last),
```

```
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);
```

```
SELECT * FROM person;
```

```
+-----+-----+
| id | name                |
+-----+-----+
|  1 | Antonio Paz         |
|  2 | Lilliana Angelovska |
+-----+-----+
```

```
SELECT * FROM shirt;
```

```
+-----+-----+-----+-----+
| id | style  | color  | owner |
+-----+-----+-----+-----+
|  1 | polo   | blue   | 1     |
|  2 | dress  | white  | 1     |
|  3 | t-shirt| blue   | 1     |
|  4 | dress  | orange | 2     |
|  5 | polo   | red    | 2     |
|  6 | dress  | blue   | 2     |
|  7 | t-shirt| white  | 2     |
+-----+-----+-----+-----+
```

```
SELECT s.* FROM person p INNER JOIN shirt s
      ON s.owner = p.id
      WHERE p.name LIKE 'Lilliana%'
      AND s.color <> 'white';
```

```
+-----+-----+-----+-----+
| id | style  | color  | owner |
+-----+-----+-----+-----+
|  4 | dress  | orange | 2     |
|  5 | polo   | red    | 2     |
|  6 | dress  | blue   | 2     |
+-----+-----+-----+-----+
```

When used in this fashion, the REFERENCES clause is not displayed in the output of SHOW CREATE TABLE or DESCRIBE:

```
SHOW CREATE TABLE shirt\G
***** 1. row *****
Table: shirt
Create Table: CREATE TABLE `shirt` (
  `id` smallint(5) unsigned NOT NULL auto_increment,
  `style` enum('t-shirt','polo','dress') NOT NULL,
  `color` enum('red','blue','orange','white','black') NOT NULL,
```

```
`owner` smallint(5) unsigned NOT NULL,  
PRIMARY KEY (`id`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

The use of REFERENCES in this way as a comment or “reminder” in a column definition works with both MyISAM and BerkeleyDB tables.

3.6.7. Searching on Two Keys

An OR using a single key is well optimized, as is the handling of AND.

The one tricky case is that of searching on two different keys combined with OR:

```
SELECT field1_index, field2_index FROM test_table  
WHERE field1_index = '1' OR field2_index = '1'
```

This case is optimized from MySQL 5.0.0. See [Section 7.2.6, “Index Merge Optimization”](#).

You can also solve the problem efficiently by using a UNION that combines the output of two separate SELECT statements. See [Section 13.2.7.2, “UNION Syntax”](#).

Each SELECT searches only one key and can be optimized:

```
SELECT field1_index, field2_index  
FROM test_table WHERE field1_index = '1'  
UNION  
SELECT field1_index, field2_index  
FROM test_table WHERE field2_index = '1';
```

3.6.8. Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,  
day INT(2) UNSIGNED ZEROFILL);  
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),  
(2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
GROUP BY year,month;
```

Which returns:

```
+-----+-----+-----+
| year | month | days |
+-----+-----+-----+
| 2000 | 01    | 3    |
| 2000 | 02    | 2    |
+-----+-----+-----+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9. Using AUTO_INCREMENT

The AUTO_INCREMENT attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (id)
);
```

```
INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');
```

```
SELECT * FROM animals;
```

Which returns:

```
+-----+-----+
| id | name      |
+-----+-----+
| 1  | dog       |
| 2  | cat       |
| 3  | penguin   |
| 4  | lax       |
| 5  | whale     |
| 6  | ostrich   |
+-----+-----+
```

You can retrieve the most recent AUTO_INCREMENT value with the

LAST_INSERT_ID() SQL function or the mysql_insert_id() C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Note: For a multiple-row insert, LAST_INSERT_ID() and mysql_insert_id() actually return the AUTO_INCREMENT key from the *first* of the inserted rows. This allows multiple-row inserts to be reproduced correctly on other servers in a replication setup.

For MyISAM and BDB tables you can specify AUTO_INCREMENT on a secondary column in a multiple-column index. In this case, the generated value for the AUTO_INCREMENT column is calculated as MAX(auto_increment_column) + 1 WHERE prefix=*given-prefix*. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (  
    grp ENUM('fish','mammal','bird') NOT NULL,  
    id MEDIUMINT NOT NULL AUTO_INCREMENT,  
    name CHAR(30) NOT NULL,  
    PRIMARY KEY (grp,id)  
);  
  
INSERT INTO animals (grp,name) VALUES  
    ('mammal','dog'),('mammal','cat'),  
    ('bird','penguin'),('fish','lax'),('mammal','whale'),  
    ('bird','ostrich');  
  
SELECT * FROM animals ORDER BY grp,id;
```

Which returns:

```
+-----+-----+-----+  
| grp   | id  | name   |  
+-----+-----+-----+  
| fish  | 1   | lax    |  
| mammal| 1   | dog    |  
| mammal| 2   | cat    |  
| mammal| 3   | whale  |  
| bird  | 1   | penguin|  
| bird  | 2   | ostrich|  
+-----+-----+-----+
```

Note that in this case (when the AUTO_INCREMENT column is part of a multiple-column index), AUTO_INCREMENT values are reused if you delete the row with the

biggest `AUTO_INCREMENT` value in any group. This happens even for MyISAM tables, for which `AUTO_INCREMENT` values normally are not reused.

If the `AUTO_INCREMENT` column is part of multiple indexes, MySQL will generate sequence values using the index that begins with the `AUTO_INCREMENT` column, if there is one. For example, if the `animals` table contained indexes `PRIMARY KEY (grp, id)` and `INDEX (id)`, MySQL would ignore the `PRIMARY KEY` for generating sequence values. As a result, the table would contain a single sequence, not a sequence per `grp` value.

To start with an `AUTO_INCREMENT` value other than 1, you can set that value with `CREATE TABLE` or `ALTER TABLE`, like this:

```
mysql> ALTER TABLE tbl1 AUTO_INCREMENT = 100;
```

More information about `AUTO_INCREMENT` is available here:

- How to assign the `AUTO_INCREMENT` attribute to a column: [Section 13.1.5, “CREATE TABLE Syntax”](#), and [Section 13.1.2, “ALTER TABLE Syntax”](#).
- How `AUTO_INCREMENT` behaves depending on the SQL mode: [Section 5.2.5, “The Server SQL Mode”](#).
- Find the row that contains the most recent `AUTO_INCREMENT` value: [Section 12.1.3, “Comparison Functions and Operators”](#).
- Set the `AUTO_INCREMENT` value to be used: [Section 13.5.3, “SET Syntax”](#).
- `AUTO_INCREMENT` and replication: [Section 6.7, “Replication Features and Known Problems”](#).
- Server-system variables related to `AUTO_INCREMENT` (`auto_increment_increment` and `auto_increment_offset`) that can be used for replication: [Section 5.2.2, “Server System Variables”](#).

3.7. Queries from the Twin Project

At Analytikerna and Lentus, we have been doing the systems and field work for a big research project. This project is a collaboration between the Institute of Environmental Medicine at Karolinska Institutet Stockholm and the Section on Clinical Research in Aging and Psychology at the University of Southern California.

The project involves a screening part where all twins in Sweden older than 65 years are interviewed by telephone. Twins who meet certain criteria are passed on to the next stage. In this latter stage, twins who want to participate are visited by a doctor/nurse team. Some of the examinations include physical and neuropsychological examination, laboratory testing, neuroimaging, psychological status assessment, and family history collection. In addition, data are collected on medical and environmental risk factors.

More information about Twin studies can be found at:

http://www.mep.ki.se/twinreg/index_en.html

The latter part of the project is administered with a Web interface written using Perl and MySQL.

Each night all data from the interviews is moved into a MySQL database.

3.7.1. Find All Non-distributed Twins

The following query is used to determine who goes into the second part of the project:

```
SELECT
  CONCAT(p1.id, p1.tvab) + 0 AS tvid,
  CONCAT(p1.christian_name, ' ', p1.surname) AS Name,
  p1.postal_code AS Code,
  p1.city AS City,
  pg.abrev AS Area,
  IF(td.participation = 'Aborted', 'A', ' ') AS A,
  p1.dead AS dead1,
  l.event AS event1,
  td.suspect AS tsuspect1,
  id.suspect AS isuspect1,
```

```

td.severe AS tsevere1,
id.severe AS isevere1,
p2.dead AS dead2,
l2.event AS event2,
h2.nurse AS nurse2,
h2.doctor AS doctor2,
td2.suspect AS tsuspect2,
id2.suspect AS isuspect2,
td2.severe AS tsevere2,
id2.severe AS isevere2,
l.finish_date
FROM
twin_project AS tp
/* For Twin 1 */
LEFT JOIN twin_data AS td ON tp.id = td.id
      AND tp.tvab = td.tvab
LEFT JOIN informant_data AS id ON tp.id = id.id
      AND tp.tvab = id.tvab
LEFT JOIN harmony AS h ON tp.id = h.id
      AND tp.tvab = h.tvab
LEFT JOIN lentus AS l ON tp.id = l.id
      AND tp.tvab = l.tvab
/* For Twin 2 */
LEFT JOIN twin_data AS td2 ON p2.id = td2.id
      AND p2.tvab = td2.tvab
LEFT JOIN informant_data AS id2 ON p2.id = id2.id
      AND p2.tvab = id2.tvab
LEFT JOIN harmony AS h2 ON p2.id = h2.id
      AND p2.tvab = h2.tvab
LEFT JOIN lentus AS l2 ON p2.id = l2.id
      AND p2.tvab = l2.tvab,
person_data AS p1,
person_data AS p2,
postal_groups AS pg
WHERE
/* p1 gets main twin and p2 gets his/her twin. */
/* ptvab is a field inverted from tvab */
p1.id = tp.id AND p1.tvab = tp.tvab AND
p2.id = p1.id AND p2.ptvab = p1.tvab AND
/* Just the screening survey */
tp.survey_no = 5 AND
/* Skip if partner died before 65 but allow emigration (dead=9)
(p2.dead = 0 OR p2.dead = 9 OR
(p2.dead = 1 AND
(p2.death_date = 0 OR
(((TO_DAYS(p2.death_date) - TO_DAYS(p2.birthday)) / 365)
>= 65))))
AND
(
/* Twin is suspect */

```

```

(td.future_contact = 'Yes' AND td.suspect = 2) OR
/* Twin is suspect - Informant is Blessed */
(td.future_contact = 'Yes' AND td.suspect = 1
    AND id.suspect = 1) OR
/* No twin - Informant is Blessed */
(ISNULL(td.suspect) AND id.suspect = 1
    AND id.future_contact = 'Yes') OR
/* Twin broken off - Informant is Blessed */
(td.participation = 'Aborted'
    AND id.suspect = 1 AND id.future_contact = 'Yes') OR
/* Twin broken off - No inform - Have partner */
(td.participation = 'Aborted' AND ISNULL(id.suspect)
    AND p2.dead = 0))

AND
l.event = 'Finished'
/* Get at area code */
AND SUBSTRING(p1.postal_code, 1, 2) = pg.code
/* Not already distributed */
AND (h.nurse IS NULL OR h.nurse=00 OR h.doctor=00)
/* Has not refused or been aborted */
AND NOT (h.status = 'Refused' OR h.status = 'Aborted'
    OR h.status = 'Died' OR h.status = 'Other')
ORDER BY
    tvid;

```

Some explanations:

- `CONCAT(p1.id, p1.tvab) + 0 AS tvid`

We want to sort on the concatenated `id` and `tvab` in numerical order. Adding `0` to the result causes MySQL to treat the result as a number.

- column `id`

This identifies a pair of twins. It is a key in all tables.

- column `tvab`

This identifies a twin in a pair. It has a value of 1 or 2.

- column `ptvab`

This is an inverse of `tvab`. When `tvab` is 1 this is 2, and vice versa. It exists to save typing and to make it easier for MySQL to optimize the query.

This query demonstrates, among other things, how to do lookups on a table from the same table with a join (p1 and p2). In the example, this is used to check whether a twin's partner died before the age of 65. If so, the row is not returned.

All of the above exist in all tables with twin-related information. We have a key on both id, tvab (all tables), and id, ptvab (person_data) to make queries faster.

On our production machine (A 200MHz UltraSPARC), this query returns about 150-200 rows and takes less than one second.

The current number of records in the tables used in the query:

Table	Rows
person_data	71074
lentus	5291
twin_project	5286
twin_data	2012
informant_data	663
harmony	381
postal_groups	100

3.7.2. Show a Table of Twin Pair Status

Each interview ends with a status code called event. The query shown here is used to display a table over all twin pairs combined by event. This indicates in how many pairs both twins are finished, in how many pairs one twin is finished and the other refused, and so on.

```
SELECT
    t1.event,
    t2.event,
    COUNT(*)
FROM
    lentus AS t1,
    lentus AS t2,
    twin_project AS tp
WHERE
    /* We are looking at one pair at a time */
```

```
t1.id = tp.id
AND t1.tvab=tp.tvab
AND t1.id = t2.id
/* Just the screening survey */
AND tp.survey_no = 5
/* This makes each pair only appear once */
AND t1.tvab='1' AND t2.tvab='2'
GROUP BY
t1.event, t2.event;
```

3.8. Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
    "%h",%Y%m%d%H%M%S}t,%>s,"%b",\%{Content-Type}o", \
    "%U",\%{Referer}i",\%{User-Agent}i"
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the LogFormat line writes to the log file.

Chapter 4. Using MySQL Programs

Table of Contents

[4.1. Overview of MySQL Programs](#)

[4.2. Invoking MySQL Programs](#)

[4.3. Specifying Program Options](#)

[4.3.1. Using Options on the Command Line](#)

[4.3.2. Using Option Files](#)

[4.3.3. Using Environment Variables to Specify Options](#)

[4.3.4. Using Options to Set Program Variables](#)

This chapter provides a brief overview of the command-line programs provided by MySQL AB and discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Later chapters provide more detailed descriptions of individual programs, including which options they recognize.

MySQL AB also provides three GUI client programs for use with MySQL Server:

- **MySQL Administrator:** This tool is used for administering MySQL servers, databases, tables, and user accounts.
- **MySQL Query Browser:** This graphical tool is provided by MySQL AB for creating, executing, and optimizing queries on MySQL databases.
- **MySQL Migration Toolkit:** This tool helps you migrate schemas and data from other relational database management systems for use with MySQL.

These GUI programs each have their own manuals that you can access at <http://dev.mysql.com/doc/>.

4.1. Overview of MySQL Programs

MySQL AB provides several types of programs:

- The MySQL server and server startup scripts:
 - **mysqld** is the MySQL server.
 - **mysqld_safe**, **mysql.server**, and **mysqld_multi** are server startup scripts.
 - **mysql_install_db** initializes the data directory and the initial databases.
 - MySQL Instance Manager monitors and manages MySQL Server instances.

[Chapter 5, *Database Administration*](#), discusses these programs further

- Client programs that access the server:
 - **mysql** is a command-line client for executing SQL statements interactively or in batch mode.
 - **mysqladmin** is an administrative client.
 - **mysqlcheck** performs table maintenance operations.
 - **mysqldump** and **mysqlhotcopy** make database backups.
 - **mysqlimport** imports data files.
 - **mysqlshow** displays information about databases and tables.

[Chapter 8, *Client and Utility Programs*](#), discusses these programs further

- Utility programs that operate independently of the server:
 - **myisamchk** performs table maintenance operations.

- **mysampack** produces compressed, read-only tables.
- **mysqlbinlog** is a tool for processing binary log files.
- **perror** displays the meaning of error codes.

[Chapter 8, *Client and Utility Programs*](#), discusses these programs further

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see [Chapter 2, *Installing and Upgrading MySQL*](#), for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install something else.

4.2. Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. “shell>” represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are \$ for **sh** or **bash**, % for **csh** or **tcsh**, and C:\> for the Windows **command.com** or **cmd.exe** command interpreters.

```
shell> mysql -u root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump --user=root personnel
```

Arguments that begin with a single or double dash (‘-’, ‘--’) are option arguments. Options typically specify the type of connection a program should make to the server or affect its operational mode. Option syntax is described in [Section 4.3, “Specifying Program Options”](#).

Non-option arguments (arguments with no leading dash) provide additional information to the program. For example, the **mysql** program interprets the first non-option argument as a database name, so the command `mysql -u root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program understands and describe the meaning of any additional non-option arguments.

Some options are common to a number of programs. The most common of these are the `--host` (or `-h`), `--user` (or `-u`), and `--password` (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the username and password of your MySQL account. All MySQL client programs understand these options; they allow you to specify which server to connect to and the account to use on that server.

Other connection options are `--port` (or `-P`) to specify a TCP/IP port number and `--socket` (or `-S`) to specify a Unix socket file on Unix (or named pipe name

on Windows).

The default hostname is `localhost`. For client programs on Unix, the hostname `localhost` is special. It causes the client to connect to the MySQL server through a Unix socket file. This occurs even if a `--port` or `-P` option is given to specify a port number. To ensure that the client makes a TCP/IP connection to the local server, use `--host` or `-h` to specify a hostname value of `127.0.0.1`, or the IP address or name of the local server. You can also specify the connection protocol explicitly, even for `localhost`, by using the `--protocol=tcp` option.

You may find it necessary to invoke MySQL programs using the pathname to the `bin` directory in which they are installed. This is likely to be the case if you get a “program not found” error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the pathname of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire pathname. For example, if **mysql** is installed in `/usr/local/mysql/bin`, you'll be able to run it by invoking it as **mysql**; it will not be necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in [Section 4.3.3, “Using Environment Variables to Specify Options”](#).)

4.3. Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is most common for options that apply to a specific invocation of the program.
- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables. This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose. However, [Section 5.13.2, “Running Multiple Servers on Unix”](#), discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for both the server and client programs.

MySQL programs determine which options are given first by examining environment variables, then by reading option files, and then by checking the command line. This means that environment variables have the lowest precedence and command-line options the highest.

Because options are processed in order, if an option is specified multiple times, the last occurrence takes precedence. The following command causes **mysql** to connect to the server running on localhost:

```
shell> mysql -h example.com -h localhost
```

If conflicting or related options are given, later options take precedence over earlier options. The following command runs **mysql** in “no column names” mode:

```
shell> mysql --column-names --skip-column-names
```

An option can be specified by writing it in full or as any unambiguous prefix. For example, the **--compress** option can be given to **mysqldump** as **--compr**, but not as **--comp** because that is ambiguous:

```
shell> mysqldump --comp
mysqldump: ambiguous option '--comp' (compatible, compress)
```

Be aware that the use of option prefixes can cause problems in the event that new options are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

You can take advantage of the way that MySQL programs process options by specifying default values for a program's options in an option file. That enables you to avoid typing them each time you run the program, but also allows you to override the defaults if necessary by using command-line options.

4.3.1. Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it has a short name or a long name. Many options have both forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.
- Option names are case sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)
- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an '=' sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=pass_val` or as `--password`. In the latter case (with no password value given), the program prompts you for the password. The password option also may be given in short form as `-ppass_val` or as `-`

p. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*. The reason for this is that if a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

```
shell> mysql -ptest
shell> mysql -p test
```

The first command instructs **mysql** to use a password value of `test`, but specifies no default database. The second instructs **mysql** to prompt for the password value and to use `test` as the default database.

Some options control behavior that can be turned on or off. For example, the **mysql** client supports a `--column-names` option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of **mysql** into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

Another option that may occasionally be useful with **mysql** is the `--execute` or `-e` option, which can be used to pass SQL statements to the server. The statements must be enclosed by single or double quotation marks. If you wish to use quoted values within a statement, you should use double quotes for the statement, and single quotes for any quoted values within the statement. When this option is used, **mysql** executes the statements and exits.

For example, you can use the following command to obtain a list of user accounts:

```
shell> mysql -u root -p --execute="SELECT User, Host FROM user" mysql
Enter password: *****
+-----+-----+
| User  | Host          |
+-----+-----+
|      | gigan        |
| root  | gigan        |
|      | localhost    |
| jon   | localhost    |
| root  | localhost    |
+-----+-----+
shell>
```

Note that the long form (`--execute`) is followed by an equals sign (=).

In the preceding example, the name of the `mysql` database was passed as a separate argument. However, the same statement could have been executed using this command, which specifies no default database:

```
mysql> mysql -u root -p --execute="SELECT User, Host FROM mysql.user"
```

Multiple SQL statements may be passed on the command line, separated by semicolons:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: *****
```

```

+-----+
| VERSION() |
+-----+
| 5.0.19-log |
+-----+
+-----+
| NOW() |
+-----+
| 2006-01-05 21:19:04 |
+-----+

```

The `--execute` or `-e` option may also be used to pass commands in an analogous fashion to the **ndb_mgm** management client for MySQL Cluster. See [Section 15.3.6, “Safe Shutdown and Restart”](#), for an example.

4.3.2. Using Option Files

Most MySQL programs can read startup options from option files (also sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program. For the MySQL server, MySQL provides a number of [preconfigured option files](#).

To determine whether a program reads option files, invoke it with the `--help` option (`--verbose` and `--help` for **mysqld**). If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

Note: Option files used with MySQL Cluster programs are covered in [Section 15.4, “MySQL Cluster Configuration”](#).

On Windows, MySQL programs read startup options from the following files:

Filename	Purpose
WINDIR\my.ini	Global options
C:\my.cnf	Global options
INSTALLDIR\my.ini	Global Options
defaults-extra-file	The file specified with <code>--defaults-extra-file=path</code> , if any

WINDIR represents the location of your Windows directory. This is commonly `C:\WINDOWS` or `C:\WINNT`. You can determine its exact location from the value of the *WINDIR* environment variable using the following command:

```
C:\> echo %WINDIR%
```

INSTALLDIR represents the installation directory of MySQL. This is typically `C:\PROGRAMDIR\MySQL\MySQL 5.0 Server` where *PROGRAMDIR* represents the programs directory (usually Program Files on English-language versions of Windows), when MySQL 5.0 has been installed using the installation and configuration wizards. See [Section 2.3.4.14, “The Location of the my.ini File”](#).

On Unix, MySQL programs read startup options from the following files:

Filename	Purpose
<code>/etc/my.cnf</code>	Global options
<code>\$MYSQL_HOME/my.cnf</code>	Server-specific options
<code>defaults-extra-file</code>	The file specified with <code>--defaults-extra-file=path</code> , if any
<code>~/.my.cnf</code>	User-specific options

MYSQL_HOME is an environment variable containing the path to the directory in which the server-specific `my.cnf` file resides. (This was *DATADIR* prior to MySQL version 5.0.3.)

If *MYSQL_HOME* is not set and you start the server using the **mysqld_safe** program, **mysqld_safe** attempts to set *MYSQL_HOME* as follows:

- Let *BASEDIR* and *DATADIR* represent the pathnames of the MySQL base directory and data directory, respectively.
- If there is a `my.cnf` file in *DATADIR* but not in *BASEDIR*, **mysqld_safe** sets *MYSQL_HOME* to *DATADIR*.
- Otherwise, if *MYSQL_HOME* is not set and there is no `my.cnf` file in *DATADIR*, **mysqld_safe** sets *MYSQL_HOME* to *BASEDIR*.

In MySQL, use of *DATADIR* as the location for `my.cnf` is deprecated. *BASEDIR* is a better location.

Typically, *DATADIR* is `/usr/local/mysql/data` for a binary installation or `/usr/local/var` for a source installation. Note that this is the data directory location that was specified at configuration time, not the one specified with the `--datadir` option when **mysqld** starts. Use of `--datadir` at runtime has no effect on where the server looks for option files, because it looks for them before processing any options.

MySQL looks for option files in the order just described and reads any that exist. If an option file that you want to use does not exist, create it with a plain text editor.

If multiple instances of a given option are found, the last instance takes precedence. There is one exception: For **mysqld**, the *first* instance of the `--user` option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Note: On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional, and acts as a security measure.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the `--help` option.

The syntax for specifying options in an option file is similar to command-line syntax, except that you omit the leading two dashes. For example, `--quick` or `--host=localhost` on the command line should be specified as `quick` or `host=localhost` in an option file. To specify an option of the form `--loose-opt_name` in an option file, write it as `loose-opt_name`.

Empty lines in option files are ignored. Non-empty lines can take any of the following forms:

- `#comment, ;comment`

Comment lines start with `'#'` or `';'`. A `'#'` comment can start in the middle of a line as well.

- `[group]`

group is the name of the program or group for which you want to set

options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given.

- `opt_name`

This is equivalent to `--opt_name` on the command line.

- `opt_name=value`

This is equivalent to `--opt_name=value` on the command line. In an option file, you can have spaces around the '=' character, something that is not true on the command line. You can enclose the value within single quotes or double quotes, which is useful if the value contains a '#' comment character or whitespace.

For options that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024² or 1024³. For example, the following command tells **mysqladmin** to ping the server 1024 times, sleeping 10 seconds between each ping:

```
mysql> mysqladmin --count=1K --sleep=10 ping
```

Leading and trailing blanks are automatically deleted from option names and values. You may use the escape sequences '\b', '\t', '\n', '\r', '\\', and '\s' in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters.

Because the '\\ ' escape sequence represents a single backslash, you must write each '\' as '\\ '. Alternatively, you can specify the value using '/' rather than '\' as the pathname separator.

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the `[mysqld]` and `[mysql]` groups apply to the **mysqld** server and the **mysql** client program, respectively.

The `[client]` option group is read by all client programs (but *not* by **mysqld**). This allows you to specify options that apply to all clients. For example, `[client]` is the perfect group to use to specify the password that you use to connect to the server. (But make sure that the option file is readable and writable only by yourself, so that other people cannot find out your password.) Be sure

not to put an option in the [client] group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock
```

```
[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M
```

```
[mysqldump]
quick
```

The preceding option file uses `var_name=value` syntax for the lines that set the `key_buffer_size` and `max_allowed_packet` variables.

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"
```

```
[mysql]
no-auto-rehash
connect_timeout=2
```

```
[mysqlhotcopy]
interactive-timeout
```

If you want to create option groups that should be read by **mysqld** servers from a specific MySQL release series only, you can do this by using groups with names of [mysqld-4.1], [mysqld-5.0], and so forth. The following group indicates that the `--new` option should be used only by MySQL servers with 5.0.x version numbers:

```
[mysqld-5.0]
new
```

Beginning with MySQL 5.0.4, it is possible to use `!include` directives in option

files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, you can use the following directive:

```
!include /home/me/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, you would use this directive:

```
!includedir /home/mydir
```

Note: Currently, any files to be found and included using the `!includedir` directive on Unix operating systems *must* have filenames ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Note that options read from included files are applied in the context of the current option group. Suppose that you were to write the following lines in `my.cnf`:

```
[mysqld]
!include /home/mydir/myopt.cnf
```

In this case, the `myopt.cnf` file is processed only for the server, and the `!include` directive is ignored by any client applications. However, if you were to use the following lines, the directory `/home/mydir/my-dump-options` is checked for option files by **mysqldump** only, and not by the server or by any other client applications:

```
[mysqldump]
!includedir /home/mydir/my-dump-options
```

If you have a source distribution, you can find sample option files named `my-xxxx.cnf` in the `support-files` directory. If you have a binary distribution, look in the `support-files` directory under your MySQL installation directory. On Windows, the sample option files may be located in the MySQL installation directory (see earlier in this section or [Chapter 2, Installing and Upgrading MySQL](#), if you do not know where this is). Currently, there are sample option files for small, medium, large, and very large systems. To experiment with one of these files, copy it to `C:\my.cnf` on Windows or to `.my.cnf` in your home directory on Unix.

Note: On Windows, the `.cnf` option file extension might not be displayed.

All MySQL programs that support option files handle the following options. They affect option-file handling, so they must be given on the command line and not in an option file. To work properly, each of these options must immediately follow the command name, with the exception that `--print-defaults` may be used immediately after `--defaults-file` or `--defaults-extra-file`.

- `--no-defaults`

Don't read any option files.

- `--print-defaults`

Print the program name and all options that it gets from option files.

- `--defaults-file=file_name`

Use only the given option file. *file_name* is the full pathname to the file.

- `--defaults-extra-file=file_name`

Read this option file after the global option file but (on Unix) before the user option file. *file_name* is the full pathname to the file. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, the program will exit with an error.

- `--defaults-group-suffix=str`

If this option is given, the program reads not only its usual option groups, but also groups with the usual names and a suffix of *str*. For example, the **mysql** client normally reads the `[client]` and `[mysql]` groups. If the `--default-group-suffix=_other` option is given, **mysql** also reads the `[client_other]` and `[mysql_other]` groups. This option was added in MySQL 5.0.10.

In shell scripts, you can use the **my_print_defaults** program to parse option files and see what options would be used by a given program. The following example shows the output that **my_print_defaults** might produce when asked to show the options found in the `[client]` and `[mysql]` groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

4.3.2.1. Preconfigured Option Files

MySQL provides a number of preconfigured option files that can be used as a basis for tuning the MySQL server. Look in your installation directory for files such as `my-small.cnf`, `my-medium.cnf`, `my-large.cnf`, and `my-huge.cnf`, which you can rename and copy to the appropriate location for use as a base configuration file. Regarding names and appropriate location, see the general information provided in [Section 4.3.2, “Using Option Files”](#). On Windows, those files have a `.ini` rather than a `.cnf` extension.

4.3.3. Using Environment Variables to Specify Options

To specify an option using an environment variable, set the variable using the syntax appropriate for your command processor. For example, on Windows or NetWare, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for `sh`, `bash`, `zsh`, and so on) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For **csh** and **tcsh**, use **setenv** to make the shell variable available to the environment:

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, place the appropriate command or commands in a startup file that your command interpreter reads each time it starts. Typical startup files are `AUTOEXEC.BAT` for Windows, `.bash_profile` for **bash**, or `.tcshrc` for **tcsh**. Consult the documentation for your command interpreter for specific details.

[Appendix F, *Environment Variables*](#), lists all environment variables that affect MySQL program operation.

4.3.4. Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime. Program variables are set the same way as any other long option that takes a value. For example, **mysql** has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for **mysql** to a value of 16MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024 , 1024^2 or 1024^3 . (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.)

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M
```

```
[mysqld]
key-buffer-size=512M
```

Note: Before MySQL 4.0.2, the only syntax for setting program variables was `--set-variable=option=value` (or `set-variable=option=value` in option files). This syntax still is recognized, but is deprecated as of MySQL 4.0.2.

Many server system variables can also be set at runtime. For details, see [Section 5.2.3.2, “Dynamic System Variables”](#).

Chapter 5. Database Administration

Table of Contents

- [5.1. Overview of Server-Side Programs](#)
- [5.2. **mysqld** — The MySQL Server](#)
 - [5.2.1. **mysqld** Command Options](#)
 - [5.2.2. Server System Variables](#)
 - [5.2.3. Using System Variables](#)
 - [5.2.4. Server Status Variables](#)
 - [5.2.5. The Server SQL Mode](#)
 - [5.2.6. The MySQL Server Shutdown Process](#)
 - [5.2.7. MySQL Server-Side Help Support](#)
- [5.3. The **mysqld-max** Extended MySQL Server](#)
- [5.4. MySQL Server Startup Programs](#)
 - [5.4.1. **mysqld_safe** — MySQL Server Startup Script](#)
 - [5.4.2. **mysql.server** — MySQL Server Startup Script](#)
 - [5.4.3. **mysqld_multi** — Manage Multiple MySQL Servers](#)
- [5.5. **mysqlmanager** — The MySQL Instance Manager](#)
 - [5.5.1. Starting the MySQL Server with MySQL Instance Manager](#)
 - [5.5.2. Connecting to the MySQL Instance Manager and Creating User Accounts](#)
 - [5.5.3. MySQL Instance Manager Command Options](#)
 - [5.5.4. MySQL Instance Manager Configuration Files](#)
 - [5.5.5. Commands Recognized by the MySQL Instance Manager](#)
- [5.6. Installation-Related Programs](#)
 - [5.6.1. **mysql_fix_privilege_tables** — Upgrade MySQL System Tables](#)
 - [5.6.2. **mysql_upgrade** — Check Tables for MySQL Upgrade](#)
- [5.7. General Security Issues](#)
 - [5.7.1. General Security Guidelines](#)
 - [5.7.2. Making MySQL Secure Against Attackers](#)
 - [5.7.3. Security-Related **mysqld** Options](#)
 - [5.7.4. Security Issues with `LOAD DATA LOCAL`](#)
 - [5.7.5. How to Run MySQL as a Normal User](#)
- [5.8. The MySQL Access Privilege System](#)
 - [5.8.1. What the Privilege System Does](#)
 - [5.8.2. How the Privilege System Works](#)

- [5.8.3. Privileges Provided by MySQL](#)
- [5.8.4. Connecting to the MySQL Server](#)
- [5.8.5. Access Control, Stage 1: Connection Verification](#)
- [5.8.6. Access Control, Stage 2: Request Verification](#)
- [5.8.7. When Privilege Changes Take Effect](#)
- [5.8.8. Causes of Access denied Errors](#)
- [5.8.9. Password Hashing as of MySQL 4.1](#)
- [5.9. MySQL User Account Management](#)
 - [5.9.1. MySQL Usernames and Passwords](#)
 - [5.9.2. Adding New User Accounts to MySQL](#)
 - [5.9.3. Removing User Accounts from MySQL](#)
 - [5.9.4. Limiting Account Resources](#)
 - [5.9.5. Assigning Account Passwords](#)
 - [5.9.6. Keeping Your Password Secure](#)
 - [5.9.7. Using Secure Connections](#)
- [5.10. Backup and Recovery](#)
 - [5.10.1. Database Backups](#)
 - [5.10.2. Example Backup and Recovery Strategy](#)
 - [5.10.3. Point-in-Time Recovery](#)
 - [5.10.4. Table Maintenance and Crash Recovery](#)
- [5.11. MySQL Localization and International Usage](#)
 - [5.11.1. The Character Set Used for Data and Sorting](#)
 - [5.11.2. Setting the Error Message Language](#)
 - [5.11.3. Adding a New Character Set](#)
 - [5.11.4. The Character Definition Arrays](#)
 - [5.11.5. String Collating Support](#)
 - [5.11.6. Multi-Byte Character Support](#)
 - [5.11.7. Problems With Character Sets](#)
 - [5.11.8. MySQL Server Time Zone Support](#)
- [5.12. MySQL Server Logs](#)
 - [5.12.1. The Error Log](#)
 - [5.12.2. The General Query Log](#)
 - [5.12.3. The Binary Log](#)
 - [5.12.4. The Slow Query Log](#)
 - [5.12.5. Server Log Maintenance](#)
- [5.13. Running Multiple MySQL Servers on the Same Machine](#)
 - [5.13.1. Running Multiple Servers on Windows](#)
 - [5.13.2. Running Multiple Servers on Unix](#)
 - [5.13.3. Using Client Programs in a Multiple-Server Environment](#)

[5.14. The MySQL Query Cache](#)

[5.14.1. How the Query Cache Operates](#)

[5.14.2. Query Cache SELECT Options](#)

[5.14.3. Query Cache Configuration](#)

[5.14.4. Query Cache Status and Maintenance](#)

This chapter covers topics that deal with administering a MySQL installation:

- Configuring the server
- Managing user accounts
- Performing backups
- The server log files
- The query cache

5.1. Overview of Server-Side Programs

The MySQL server, **mysqld**, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that perform setup operations when you install MySQL or that assist you in starting and stopping the server. This section provides an overview of the server and related programs. The following sections provide more detailed information about each of these programs.

Each MySQL program takes many different options. Most programs provide a `-help` option that you can use to get a description of the program's different options. For example, try **mysqld --help**.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. [Section 4.3, “Specifying Program Options”](#).

The following list briefly describes the MySQL server and server-related programs:

- **mysqld**

The SQL daemon (that is, the MySQL server). To use client programs, **mysqld** must be running, because clients gain access to databases by connecting to the server. See [Section 5.2, “mysqld — The MySQL Server”](#).

- **mysqld-max**

A version of the server that includes additional features. See [Section 5.3, “The **mysqld-max** Extended MySQL Server”](#).

- **mysqld_safe**

A server startup script. **mysqld_safe** attempts to start **mysqld-max** if it exists, and **mysqld** otherwise. See [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#).

- **mysql.server**

A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes **mysqld_safe** to start the MySQL server. See [Section 5.4.2, “**mysql.server** — MySQL Server Startup Script”](#).

- **mysqld_multi**

A server startup script that can start or stop multiple servers installed on the system. See [Section 5.4.3, “**mysqld_multi** — Manage Multiple MySQL Servers”](#). As of MySQL 5.0.3 (Unix-like systems) or 5.0.13 (Windows), an alternative to **mysqld_multi** is **mysqlmanager**, the MySQL Instance Manager. See [Section 5.5, “**mysqlmanager** — The MySQL Instance Manager”](#).

- **mysql_install_db**

This script creates the MySQL database and initializes the grant tables with default privileges. It is usually executed only once, when first installing MySQL on a system. See [Section 2.10.2, “Unix Post-Installation Procedures”](#).

- **mysql_fix_privilege_tables**

This program is used after a MySQL upgrade operation. It updates the grant tables with any changes that have been made in newer versions of MySQL. See [Section 5.6.1, “**mysql_fix_privilege_tables** — Upgrade MySQL System Tables”](#).

Note: As of MySQL 5.0.19, this program has been superseded by **mysql_upgrade**.

- **mysql_upgrade**

This program is used after a MySQL upgrade operation. It checks tables for incompatibilities and repairs them if necessary, and updates the grant tables with any changes that have been made in newer versions of MySQL. See [Section 5.6.2, “**mysql_upgrade** — Check Tables for MySQL Upgrade”](#).

- **mysqlmanager**

The MySQL Instance Manager, a program for monitoring and managing MySQL servers. See [Section 5.5, “mysqlmanager — The MySQL Instance Manager”](#).

There are several other programs that are run on the server host:

- **make_binary_distribution**

This program makes a binary release of a compiled MySQL. This could be sent by FTP to /pub/mysql/upload/ on ftp.mysql.com for the convenience of other MySQL users.

5.2. **mysqld** — The MySQL Server

mysqld is the MySQL server. The following discussion covers these MySQL server configuration topics:

- Startup options that the server supports
- Server system variables
- Server status variables
- How to set the server SQL mode
- The server shutdown process

5.2.1. **mysqld** Command Options

When you start the **mysqld** server, you can specify program options using any of the methods described in [Section 4.3, “Specifying Program Options”](#). The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See [Section 4.3.2, “Using Option Files”](#).

mysqld reads options from the [mysqld] and [server] groups. **mysqld_safe** reads options from the [mysqld], [server], [mysqld_safe], and [safe_mysqld] groups. **mysql.server** reads options from the [mysqld] and [mysql.server] groups.

An embedded MySQL server usually reads options from the [server], [embedded], and [xxxxx_SERVER] groups, where xxxxx is the name of the application into which the server is embedded.

mysqld accepts many command options. For a brief summary, execute **mysqld -help**. To see the full list, use **mysqld --verbose --help**.

The following list shows some of the most common server options. Additional options are described in other sections:

- Options that affect security: See [Section 5.7.3, “Security-Related **mysqld** Options”](#).
- SSL-related options: See [Section 5.9.7.3, “SSL Command Options”](#).
- Binary log control options: See [Section 5.12.3, “The Binary Log”](#).
- Replication-related options: See [Section 6.8, “Replication Startup Options”](#).
- Options specific to particular storage engines: See [Section 14.1.1, “MyISAM Startup Options”](#), [Section 14.5.3, “BDB Startup Options”](#), [Section 14.2.4, “InnoDB Startup Options and System Variables”](#), and [Section 15.6.5.1, “MySQL Cluster-Related Command Options for **mysqld**”](#).

You can also set the values of server system variables by using variable names as options, as described later in this section.

- `--help, -?`

Display a short help message and exit. Use both the `--verbose` and `--help` options to see the full message.

- `--allow-suspicious-udfs`

This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. This option was added in version 5.0.3. See [Section 24.2.4.6, “User-Defined Function Security Precautions”](#).

- `--ansi`

Use standard (ANSI) SQL syntax instead of MySQL syntax. For more precise control over the server SQL mode, use the `--sql-mode` option instead. See [Section 1.9.3, “Running MySQL in ANSI Mode”](#), and [Section 5.2.5, “The Server SQL Mode”](#).

- `--basedir=path, -b path`

The path to the MySQL installation directory. All paths are usually resolved relative to this directory.

- `--bind-address=IP`

The IP address to bind to.

- `--bootstrap`

This option is used by the **mysql_install_db** script to create the MySQL privilege tables without having to start a full MySQL server.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--character-set-client-handshake`

Don't ignore character set information sent by the client. To ignore client information and use the default server character set, use `--skip-character-set-client-handshake`; this makes MySQL behave like MySQL 4.0.

- `--character-set-filesystem=charset_name`

The filesystem character set. This option sets the `character_set_filesystem` system variable. It was added in MySQL 5.0.19.

- `--character-set-server=charset_name, -C charset_name`

Use *charset_name* as the default server character set. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--chroot=path`

Put the **mysqld** server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure. Note that use of this option somewhat limits `LOAD DATA INFILE` and `SELECT ...`

INTO outfile.

- `--collation-server=collation_name`

Use *collation_name* as the default server collation. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--console`

(Windows only.) Write error log messages to stderr and stdout even if `--log-error` is specified. **mysqld** does not close the console window if this option is used.

- `--core-file`

Write a core file if **mysqld** dies. For some systems, you must also specify the `--core-file-size` option to **mysqld_safe**. See [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#). Note that on some systems, such as Solaris, you do not get a core file if you are also using the `--user` option.

- `--datadir=path, -h path`

The path to the data directory.

- `--debug[=debug_options], -# [debug_options]`

If MySQL is configured with `--with-debug`, you can use this option to get a trace file of what **mysqld** is doing. The *debug_options* string often is `'d:t:o,file_name'`. The default is `'d:t:i:o,mysqld.trace'`. See [Section E.1.2, “Creating Trace Files”](#).

- `--default-character-set=charset_name (DEPRECATED)`

Use *charset_name* as the default character set. This option is deprecated in favor of `--character-set-server`. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--default-collation=collation_name`

Use *collation_name* as the default collation. This option is deprecated in favor of `--collation-server`. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--default-storage-engine=type`

Set the default storage engine (table type) for tables. See [Chapter 14, *Storage Engines and Table Types*](#).

- `--default-table-type=type`

This option is a synonym for `--default-storage-engine`.

- `--default-time-zone=timezone`

Set the default server time zone. This option sets the global `time_zone` system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` system variable).

- `--delay-key-write[={OFF|ON|ALL}]`

Specify how to use delayed key writes. Delayed key writing causes key buffers not to be flushed between writes for MyISAM tables. `OFF` disables delayed key writes. `ON` enables delayed key writes for those tables that were created with the `DELAY_KEY_WRITE` option. `ALL` delays key writes for all MyISAM tables. See [Section 7.5.2, “Tuning Server Parameters”](#), and [Section 14.1.1, “MyISAM Startup Options”](#).

Note: If you set this variable to `ALL`, you should not use MyISAM tables from within another program (such as another MySQL server or `myisamchk`) when the tables are in use. Doing so leads to index corruption.

- `--des-key-file=file_name`

Read the default DES keys from this file. These keys are used by the `DES_ENCRYPT()` and `DES_DECRYPT()` functions.

- `--enable-named-pipe`

Enable support for named pipes. This option applies only on Windows NT, 2000, XP, and 2003 systems, and can be used only with the **mysqld-nt** and **mysqld-max-nt** servers that support named-pipe connections.

- `--exit-info[=flags], -T [flags]`

This is a bit mask of different flags that you can use for debugging the **mysqld** server. Do not use this option unless you know *exactly* what it does!

- `--external-locking`

Enable external locking (system locking), which is disabled by default as of MySQL 4.0. Note that if you use this option on a system on which `lockd` does not fully work (such as Linux), it is easy for **mysqld** to deadlock. This option previously was named `--enable-locking`.

Note: If you use this option to enable updates to MyISAM tables from many MySQL processes, you must ensure that the following conditions are satisfied:

- You should not use the query cache for queries that use tables that are updated by another process.
- You should not use `--delay-key-write=ALL` or `DELAY_KEY_WRITE=1` on any shared tables.

The easiest way to ensure this is to always use `--external-locking` together with `--delay-key-write=OFF` and `--query-cache-size=0`. (This is not done by default because in many setups it is useful to have a mixture of the preceding options.)

- `--flush`

Flush (synchronize) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#).

- `--init-file=file_name`

Read SQL statements from this file at startup. Each statement must be on a single line and should not include comments.

- `--innodb-safe-binlog`

Adds consistency guarantees between the content of InnoDB tables and the binary log. See [Section 5.12.3, “The Binary Log”](#). This option was removed in MySQL 5.0.3, having been made obsolete by the introduction of XA transaction support.

- `--innodb-xxx`

The InnoDB options are listed in [Section 14.2.4, “InnoDB Startup Options and System Variables”](#).

- `--language=lang_name, -L lang_name`

Return client error messages in the given language. *lang_name* can be given as the language name or as the full pathname to the directory where the language files are installed. See [Section 5.11.2, “Setting the Error Message Language”](#).

- `--large-pages`

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and OS. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

Currently, MySQL supports only the Linux implementation of large pages support (which is called HugeTLB in Linux). We have plans to extend this support to FreeBSD, Solaris and possibly other platforms.

Before large pages can be used on Linux, it is necessary to configure the HugeTLB memory pool. For reference, consult the `hugetlbpage.txt` file in the Linux kernel source.

This option is disabled by default. It was added in MySQL 5.0.3.

- `--log[=file_name], -l [file_name]`

Log connections and SQL statements received from clients to this file. See [Section 5.12.2, “The General Query Log”](#). If you omit the filename, MySQL uses `host_name.log` as the filename.

- `--log-bin[=base_name]`

Enable binary logging. The server logs all statements that change data to the binary log, which is used for backup and replication. See [Section 5.12.3, “The Binary Log”](#).

The option value, if given, is the basename for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the basename. It is recommended that you specify a basename (see [Section A.8.1, “Open Issues in MySQL”](#), for the reason). Otherwise, MySQL uses `host_name-bin` as the basename.

- `--log-bin-index[=file_name]`

The index file for binary log filenames. See [Section 5.12.3, “The Binary Log”](#). If you omit the filename, and if you didn't specify one with `--log-bin`, MySQL uses `host_name-bin.index` as the filename.

- `--log-bin-trust-function-creators[={0|1}]`

With no argument or an argument of 1, this option sets the `log_bin_trust_function_creators` system variable to 1. With an argument of 0, this option sets the system variable to 0. `log_bin_trust_function_creators` affects how MySQL enforces restrictions on stored function creation. See [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

This option was added in MySQL 5.0.16.

- `--log-bin-trust-routine-creators[={0|1}]`

This is the old name for `--log-bin-trust-function-creators`. Before MySQL 5.0.16, it also applies to stored procedures, not just stored functions and sets the `log_bin_trust_routine_creators` system variable. As of

5.0.16, this option is deprecated. It is recognized for backward compatibility but its use results in a warning.

This option was added in MySQL 5.0.6.

- `--log-error[=file_name]`

Log errors and startup messages to this file. See [Section 5.12.1, “The Error Log”](#). If you omit the filename, MySQL uses `host_name.err`. If the filename has no extension, the server adds an extension of `.err`.

- `--log-isam[=file_name]`

Log all MyISAM changes to this file (used only when debugging MyISAM).

- `--log-long-format` (*DEPRECATED*)

Log extra information to the update log, binary update log, and slow query log, if they have been activated. For example, the username and timestamp are logged for all queries. This option is deprecated, as it now represents the default logging behavior. (See the description for `--log-short-format`.) The `--log-queries-not-using-indexes` option is available for the purpose of logging queries that do not use indexes to the slow query log.

- `--log-queries-not-using-indexes`

If you are using this option with `--log-slow-queries`, queries that do not use indexes are logged to the slow query log. See [Section 5.12.4, “The Slow Query Log”](#).

- `--log-short-format`

Log less information to the update log, binary update log, and slow query log, if they have been activated. For example, the username and timestamp are not logged for queries.

- `--log-slow-admin-statements`

Log slow administrative statements such as `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `ALTER TABLE` to the slow query log.

- `--log-slow-queries[=file_name]`

Log all queries that have taken more than `long_query_time` seconds to execute to this file. See [Section 5.12.4, “The Slow Query Log”](#). See the descriptions of the `--log-long-format` and `--log-short-format` options for details.

- `--log-warnings[=level], -w [level]`

Print out warnings such as `Aborted connection...` to the error log. Enabling this option is recommended, for example, if you use replication (you get more information about what is happening, such as messages about network failures and reconnections). This option is enabled (1) by default, and the default `level` value if omitted is 1. To disable this option, use `--log-warnings=0`. Aborted connections are not logged to the error log unless the value is greater than 1. See [Section A.2.10, “Communication Errors and Aborted Connections”](#).

- `--low-priority-updates`

Give table-modifying operations (`INSERT`, `REPLACE`, `DELETE`, `UPDATE`) lower priority than selects. This can also be done via `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query, or by `SET LOW_PRIORITY_UPDATES=1` to change the priority in one thread. See [Section 7.3.2, “Table Locking Issues”](#).

- `--memlock`

Lock the **mysqld** process in memory. This works on systems such as Solaris that support the `mlockall()` system call. This might help if you have a problem where the operating system is causing **mysqld** to swap on disk. Note that use of this option requires that you run the server as root, which is normally not a good idea for security reasons. See [Section 5.7.5, “How to Run MySQL as a Normal User”](#).

- `--myisam-recover[=option[,option]...]`

Set the MyISAM storage engine recovery mode. The option value is any combination of the values of `DEFAULT`, `BACKUP`, `FORCE`, or `QUICK`. If you specify multiple values, separate them by commas. You can also use a value

of "" to disable this option. If this option is used, each time **mysqld** opens a MyISAM table, it checks whether the table is marked as crashed or wasn't closed properly. (The last option works only if you are running with external locking disabled.) If this is the case, **mysqld** runs a check on the table. If the table was corrupted, **mysqld** attempts to repair it.

The following options affect how the repair works:

Option	Description
DEFAULT	The same as not giving any option to --myisam-recover.
BACKUP	If the data file was changed during recovery, save a backup of the tbl_name.MYD file as tbl_name-datetime.BAK.
FORCE	Run recovery even if we would lose more than one row from the .MYD file.
QUICK	Don't check the rows in the table if there aren't any delete blocks.

Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options `BACKUP`, `FORCE`. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

See [Section 14.1.1, “MyISAM Startup Options”](#).

- --ndb-connectstring=connect_string

When using the NDB storage engine, it is possible to point out the management server that distributes the cluster configuration by setting the connect string option. See [Section 15.4.4.2, “The Cluster connectstring”](#), for syntax.

- --ndbcluster

If the binary includes support for the NDB Cluster storage engine, this option enables the engine, which is disabled by default. See [Chapter 15, *MySQL Cluster*](#).

- --old-passwords

Force the server to generate short (pre-4.1) password hashes for new passwords. This is useful for compatibility when the server must support older client programs. See [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#).

- `--one-thread`

Only use one thread (for debugging under Linux). This option is available only if the server is built with debugging enabled. See [Section E.1, “Debugging a MySQL Server”](#).

- `--open-files-limit=count`

Change the number of file descriptors available to **mysqld**. If this option is not set or is set to 0, **mysqld** uses the value to reserve file descriptors with `setrlimit()`. If the value is 0, **mysqld** reserves `max_connections×5` or `max_connections + table_open_cache×2` files (whichever is larger). You should try increasing this value if **mysqld** gives you the error Too many open files.

- `--pid-file=path`

The pathname of the process ID file. This file is used by other programs such as **mysqld_safe** to determine the server's process ID.

- `--port=port_num, -P port_num`

The port number to use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the root system user.

- `--port-open-timeout=num`

On some systems, when the server is stopped, the TCP/IP port might not become available immediately. If the server is restarted quickly afterward, its attempt to reopen the port can fail. This option indicates how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. The default is not to wait. This option was added in MySQL 5.0.19.

- `--safe-mode`

Skip some optimization stages.

- `--safe-show-database` (*DEPRECATED*)

See [Section 5.8.3, “Privileges Provided by MySQL”](#).

- `--safe-user-create`

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement, if the user doesn't have the `INSERT` privilege for the `mysql.user` table or any column in the table.

- `--secure-auth`

Disallow authentication by clients that attempt to use accounts that have old (pre-4.1) passwords.

- `--shared-memory`

Enable shared-memory connections by local clients. This option is available only on Windows.

- `--shared-memory-base-name=name`

The name of shared memory to use for shared-memory connections. This option is available only on Windows. The default name is `MYSQL`. The name is case sensitive.

- `--skip-bdb`

Disable the BDB storage engine. This saves memory and might speed up some operations. Do not use this option if you require BDB tables.

- `--skip-concurrent-insert`

Turn off the ability to select and insert at the same time on MyISAM tables. (This is to be used only if you think you have found a bug in this feature.) See [Section 7.3.3, “Concurrent Inserts”](#).

- `--skip-external-locking`

Do not use external locking (system locking). With external locking disabled, you must shut down the server to use **myisamchk**. (See [Section 1.4.3, “MySQL Stability”](#).) To avoid this requirement, use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair MyISAM tables.

External locking has been disabled by default since MySQL 4.0.

- `--skip-grant-tables`

This option causes the server not to use the privilege system at all, which gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing **mysqladmin flush-privileges** or **mysqladmin reload** command from a system shell, or by issuing a `MySQL FLUSH PRIVILEGES` statement after connecting to the server. This option also suppresses loading of user-defined functions (UDFs).

- `--skip-host-cache`

Do not use the internal hostname cache for faster name-to-IP resolution. Instead, query the DNS server every time a client connects. See [Section 7.5.6, “How MySQL Uses DNS”](#).

- `--skip-innodb`

Disable the InnoDB storage engine. This saves memory and disk space and might speed up some operations. Do not use this option if you require InnoDB tables.

- `--skip-merge`

Disable the MERGE storage engine. This option was added in MySQL 5.0.24. It can be used if the following behavior is undesirable: If a user has access to MyISAM table *t*, that user can create a MERGE table *m* that accesses *t*. However, if the user's privileges on *t* are subsequently revoked, the user can continue to access *t* by doing so through *m*.

- `--skip-name-resolve`

Do not resolve hostnames when checking client connections. Use only IP numbers. If you use this option, all `Host` column values in the grant tables must be IP numbers or `localhost`. See [Section 7.5.6, “How MySQL Uses DNS”](#).

- `--skip-ndbcluster`

Disable the `NDB Cluster` storage engine. This is the default for binaries that were built with `NDB Cluster` storage engine support; the server allocates memory and other resources for this storage engine only if the `--ndbcluster` option is given explicitly. See [Section 15.4.3, “Quick Test Setup of MySQL Cluster”](#), for an example of usage.

- `--skip-networking`

Don't listen for TCP/IP connections at all. All interaction with **mysqld** must be made via named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are allowed. See [Section 7.5.6, “How MySQL Uses DNS”](#).

- `--ssl*`

Options that begin with `--ssl` specify whether to allow clients to connect via SSL and indicate where to find SSL keys and certificates. See [Section 5.9.7.3, “SSL Command Options”](#).

- `--standalone`

Available on Windows NT-based systems only; instructs the MySQL server not to run as a service.

- `--symbolic-links`, `--skip-symbolic-links`

Enable or disable symbolic link support. This option has different effects on Windows and Unix:

- On Windows, enabling symbolic links allows you to establish a symbolic link to a database directory by creating a `db_name.sym` file

that contains the path to the real directory. See [Section 7.6.1.3, “Using Symbolic Links for Databases on Windows”](#).

- On Unix, enabling symbolic links means that you can link a MyISAM index file or data file to another directory with the `INDEX DIRECTORY` or `DATA DIRECTORY` options of the `CREATE TABLE` statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See [Section 7.6.1.2, “Using Symbolic Links for Tables on Unix”](#).

- `--skip-safemalloc`

If MySQL is configured with `--with-debug=full`, all MySQL programs check for memory overruns during each memory allocation and memory freeing operation. This checking is very slow, so for the server you can avoid it when you don't need it by using the `--skip-safemalloc` option.

- `--skip-show-database`

With this option, the `SHOW DATABASES` statement is allowed only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. Without this option, `SHOW DATABASES` is allowed to all users, but displays each database name only if the user has the `SHOW DATABASES` privilege or some privilege for the database. Note that *any* global privilege is considered a privilege for the database.

- `--skip-stack-trace`

Don't write stack traces. This option is useful when you are running **mysqld** under a debugger. On some systems, you also must use this option to get a core file. See [Section E.1, “Debugging a MySQL Server”](#).

- `--skip-thread-priority`

Disable using thread priorities for faster response time.

- `--socket=path`

On Unix, this option specifies the Unix socket file to use when listening for local connections. The default value is `/tmp/mysql.sock`. On Windows, the

option specifies the pipe name to use when listening for local connections that use a named pipe. The default value is MySQL (not case sensitive).

- `--sql-mode=value[,value[,value...]]`

Set the SQL mode. See [Section 5.2.5, “The Server SQL Mode”](#).

- `--sysdate-is-now`

As of MySQL 5.0.13, `SYSDATE()` by default returns the time at which it executes, not the time at which the statement in which it occurs begins executing. This differs from the behavior of `NOW()`. This option causes `SYSDATE()` to be an alias for `NOW()`. For information about the implications for binary logging and replication, see the description for `SYSDATE()` in [Section 12.5, “Date and Time Functions”](#) and for `SET TIMESTAMP` in [Section 13.5.3, “SET Syntax”](#).

This option was added in MySQL 5.0.20.

- `--temp-pool`

This option causes most temporary files created by the server to use a small set of names, rather than a unique name for each new file. This works around a problem in the Linux kernel dealing with creating many new files with different names. With the old behavior, Linux seems to “leak” memory, because it is being allocated to the directory entry cache rather than to the disk cache.

- `--transaction-isolation=level`

Sets the default transaction isolation level. The `level` value can be `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`. See [Section 13.4.6, “SET TRANSACTION Syntax”](#).

- `--tmpdir=path, -t path`

The path of the directory to use for creating temporary files. It might be useful if your default `/tmp` directory resides on a partition that is too small to hold temporary tables. This option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (‘:’) on

Unix and semicolon characters (‘;’) on Windows, NetWare, and OS/2. If the MySQL server is acting as a replication slave, you should not set `--tmpdir` to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. For more information about the storage location of temporary files, see [Section A.4.4, “Where MySQL Stores Temporary Files”](#). A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

- `--user={user_name|user_id}, -u {user_name|user_id}`

Run the **mysqld** server as the user having the name *user_name* or the numeric user ID *user_id*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

This option is *mandatory* when starting **mysqld** as root. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as root. See [Section 5.7.1, “General Security Guidelines”](#).

To avoid a possible security hole where a user adds a `--user=root` option to a `my.cnf` file (thus causing the server to run as root), **mysqld** uses only the first `--user` option specified and produces a warning if there are multiple `--user` options. Options in `/etc/my.cnf` and `$MYSQL_HOME/my.cnf` are processed before command-line options, so it is recommended that you put a `--user` option in `/etc/my.cnf` and specify a value other than root. The option in `/etc/my.cnf` is found before any other `--user` options, which ensures that the server runs as a user other than root, and that a warning results if any other `--user` option is found.

- `--version, -V`

Display version information and exit.

You can assign a value to a server system variable by using an option of the form `--var_name=value`. For example, `--key_buffer_size=32M` sets the `key_buffer_size` variable to a value of 32MB.

Note that when you assign a value to a variable, MySQL might automatically

correct the value to stay within a given range, or adjust the value to the closest allowable value if only certain values are allowed.

If you want to restrict the maximum value to which a variable can be set at runtime with SET, you can define this by using the `--maximum-var_name=value` command-line option.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

You can change the values of most system variables for a running server with the SET statement. See [Section 13.5.3, “SET Syntax”](#).

[Section 5.2.2, “Server System Variables”](#), provides a full description for all variables, and additional information for setting them at server startup and runtime. [Section 7.5.2, “Tuning Server Parameters”](#), includes information on optimizing the server by tuning system variables.

5.2.2. Server System Variables

The **mysql** server maintains many system variables that indicate how it is configured. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the SET statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

There are several ways to see the names and values of system variables:

- To see the values that a server will use based on its compiled-in defaults and any option files that it reads, use this command:

```
mysqld --verbose --help
```

- To see the values that a server will use based on its compiled-in defaults, ignoring the settings in any option files, use this command:

```
mysqld --no-defaults --verbose --help
```

- To see the current values used by a running server, use the `SHOW VARIABLES` statement.

This section provides a description of each system variable. Variables with no version indicated are present in all MySQL 5.0 releases. For historical information concerning their implementation, please see *MySQL 3.23, 4.0, 4.1 Reference Manual*.

For additional system variable information, see these sections:

- [Section 5.2.3, “Using System Variables”](#), discusses the syntax for setting and displaying system variable values.
- [Section 5.2.3.2, “Dynamic System Variables”](#), lists the variables that can be set at runtime.
- Information on tuning system variables can be found in [Section 7.5.2, “Tuning Server Parameters”](#).
- [Section 14.2.4, “InnoDB Startup Options and System Variables”](#), lists InnoDB system variables.

Note: Some of the following variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1` works but `--delay_key_write=ON` does not.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

- `auto_increment_increment`

`auto_increment_increment` and `auto_increment_offset` are intended for use with master-to-master replication, and can be used to control the operation of `AUTO_INCREMENT` columns. Both variables can be set globally or locally, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these

two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of `auto_increment_increment` or `auto_increment_offset` to a non-integer value gives rise to an error, and the actual value of the variable remains unchanged.

These two variables affect `AUTO_INCREMENT` column behavior as follows:

- `auto_increment_increment` controls the interval between successive column values. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| auto_increment_increment | 1     |
| auto_increment_offset   | 1     |
+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE autoinc1
-> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| auto_increment_increment | 10    |
| auto_increment_offset   | 1     |
+-----+-----+
2 rows in set (0.01 sec)
```

```
mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1   |
| 11  |
| 21  |
```

```
| 31 |
+-----+
4 rows in set (0.00 sec)
```

(Note how SHOW VARIABLES is used here to obtain the current values for these variables.)

- auto_increment_offset determines the starting point for the AUTO_INCREMENT column value. Consider the following, assuming that these statements are executed during the same session as the example given in the description for auto_increment_increment:

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| auto_increment_increment | 10    |
| auto_increment_offset   | 5     |
+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> CREATE TABLE autoinc2
-> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT col FROM autoinc2;
+-----+
| col |
+-----+
| 5   |
| 15  |
| 25  |
| 35  |
+-----+
4 rows in set (0.02 sec)
```

If the value of auto_increment_offset is greater than that of auto_increment_increment, the value of auto_increment_offset is ignored.

Should one or both of these variables be changed and then new rows inserted into a table containing an `AUTO_INCREMENT` column, the results may seem counterintuitive because the series of `AUTO_INCREMENT` values is calculated without regard to any values already present in the column, and the next value inserted is the least value in the series that is greater than the maximum existing value in the `AUTO_INCREMENT` column. In other words, the series is calculated like so:

$$\text{auto_increment_offset} + N \times \text{auto_increment_increment}$$

where N is a positive integer value in the series [1, 2, 3, ...]. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| auto_increment_increment | 10    |
| auto_increment_offset   | 5     |
+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1   |
| 11  |
| 21  |
| 31  |
+-----+
4 rows in set (0.00 sec)
```

```
mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL)
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
| 1   |
| 11  |
| 21  |
| 31  |
| 35  |
| 45  |
| 55  |
+-----+
```

```
| 65 |  
+-----+  
8 rows in set (0.00 sec)
```

The values shown for `auto_increment_increment` and `auto_increment_offset` generate the series $5 + N \times 10$, that is, [5, 15, 25, 35, 45, ...]. The greatest value present in the `col` column prior to the `INSERT` is 31, and the next available value in the `AUTO_INCREMENT` series is 35, so the inserted values for `col` begin at that point and the results are as shown for the `SELECT` query.

It is important to remember that it is not possible to confine the effects of these two variables to a single table, and thus they do not take the place of the sequences offered by some other database management systems; these variables control the behavior of all `AUTO_INCREMENT` columns in *all* tables on the MySQL server. If one of these variables is set globally, its effects persist until the global value is changed or overridden by setting them locally, or until **mysqld** is restarted. If set locally, the new value affects `AUTO_INCREMENT` columns for all tables into which new rows are inserted by the current user for the duration of the session, unless the values are changed during that session.

The `auto_increment_increment` variable was added in MySQL 5.0.2. Its default value is 1. See [Section 6.13, “Auto-Increment in Multiple-Master Replication”](#).

- `auto_increment_offset`

This variable was introduced in MySQL 5.0.2. Its default value is 1. For particulars, see the description for `auto_increment_increment`.

- `back_log`

The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The `back_log` value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a

short period of time.

In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix `listen()` system call should have more details. Check your OS documentation for the maximum value for this variable. `back_log` cannot be set higher than your operating system limit.

- `basedir`

The MySQL installation base directory. This variable can be set with the `--basedir` option.

- `bdb_cache_size`

The size of the buffer that is allocated for caching indexes and rows for BDB tables. If you don't use BDB tables, you should start **mysqld** with `--skip-bdb` to not allocate memory for this cache.

- `bdb_home`

The base directory for BDB tables. This should be assigned the same value as the `datadir` variable.

- `bdb_log_buffer_size`

The size of the buffer that is allocated for caching indexes and rows for BDB tables. If you don't use BDB tables, you should set this to 0 or start **mysqld** with `--skip-bdb` to not allocate memory for this cache.

- `bdb_logdir`

The directory where the BDB storage engine writes its log files. This variable can be set with the `--bdb-logdir` option.

- `bdb_max_lock`

The maximum number of locks that can be active for a BDB table (10,000 by default). You should increase this value if errors such as the following occur when you perform long transactions or when **mysqld** has to examine many

rows to calculate a query:

```
bdb: Lock table is out of available locks  
Got error 12 from ...
```

- `bdb_shared_data`

This is ON if you are using `--bdb-shared-data` to start Berkeley DB in multi-process mode. (Do not use `DB_PRIVATE` when initializing Berkeley DB.)

- `bdb_tmpdir`

The BDB temporary file directory.

- `binlog_cache_size`

The size of the cache to hold the SQL statements for the binary log during a transaction. A binary log cache is allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (`--log-bin` option). If you often use large, multiple-statement transactions, you can increase this cache size to get more performance. The `Binlog_cache_use` and `Binlog_cache_disk_use` status variables can be useful for tuning the size of this variable. See [Section 5.12.3, “The Binary Log”](#).

- `bulk_insert_buffer_size`

MyISAM uses a special tree-like cache to make bulk inserts faster for `INSERT ... SELECT`, `INSERT ... VALUES (...), (...), ...`, and `LOAD DATA INFILE` when adding data to non-empty tables. This variable limits the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. The default value is 8MB.

- `character_set_client`

The character set for statements that arrive from the client.

- `character_set_connection`

The character set used for literals that do not have a character set introducer

and for number-to-string conversion.

- `character_set_database`

The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `character_set_server`.

- `character_set_filesystem`

The filesystem character set. This variable is used to interpret string literals that refer to filenames, such as in the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. Such filenames are converted from `character_set_client` to `character_set_filesystem` before the file opening attempt occurs. The default value is `binary`, which means that no conversion occurs. For systems on which multi-byte filenames are allowed, a different value may be more appropriate. For example, if the system represents filenames using UTF-8, set `character_set_filesystem` to `'utf8'`. This variable was added in MySQL 5.0.19.

- `character_set_results`

The character set used for returning query results to the client.

- `character_set_server`

The server's default character set.

- `character_set_system`

The character set used by the server for storing identifiers. The value is always `utf8`.

- `character_sets_dir`

The directory where character sets are installed.

- `collation_connection`

The collation of the connection character set.

- `collation_database`

The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `collation_server`.

- `collation_server`

The server's default collation.

- `completion_type`

The transaction completion type:

- If the value is 0 (the default), `COMMIT` and `ROLLBACK` are unaffected.
- If the value is 1, `COMMIT` and `ROLLBACK` are equivalent to `COMMIT AND CHAIN` and `ROLLBACK AND CHAIN`, respectively. (A new transaction starts immediately with the same isolation level as the just-terminated transaction.)
- If the value is 2, `COMMIT` and `ROLLBACK` are equivalent to `COMMIT RELEASE` and `ROLLBACK RELEASE`, respectively. (The server disconnects after terminating the transaction.)

This variable was added in MySQL 5.0.3

- `concurrent_insert`

If `ON` (the default), MySQL allows `INSERT` and `SELECT` statements to run concurrently for `MyISAM` tables that have no free blocks in the middle. You can turn this option off by starting **mysqld** with `--safe` or `--skip-new`.

In MySQL 5.0.6, this variable was changed to take three integer values:

Value	Description
0	Off
1	(Default) Enables concurrent insert for <code>MyISAM</code> tables that don't

	have holes
2	Enables concurrent inserts for all MyISAM tables. If table has a hole and is in use by another thread the new row will be inserted at end of table. If table is not in use, MySQL does a normal read lock and inserts the new row into the hole.

See also [Section 7.3.3, “Concurrent Inserts”](#).

- connect_timeout

The number of seconds that the **mysqld** server waits for a connect packet before responding with Bad handshake.

- datadir

The MySQL data directory. This variable can be set with the --datadir option.

- date_format

This variable is not implemented.

- datetime_format

This variable is not implemented.

- default_week_format

The default mode value to use for the WEEK() function. See [Section 12.5, “Date and Time Functions”](#).

- delay_key_write

This option applies only to MyISAM tables. It can have one of the following values to affect handling of the DELAY_KEY_WRITE table option that can be used in CREATE TABLE statements.

Option	Description
OFF	DELAY_KEY_WRITE is ignored.

ON	MySQL honors any DELAY_KEY_WRITE option specified in CREATE TABLE statements. This is the default value.
ALL	All new opened tables are treated as if they were created with the DELAY_KEY_WRITE option enabled.

If DELAY_KEY_WRITE is enabled for a table, the key buffer is not flushed for the table on every index update, but only when the table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all MyISAM tables by starting the server with the --myisam-recover option (for example, --myisam-recover=BACKUP, FORCE). See [Section 5.2.1, “mysqld Command Options”](#), and [Section 14.1.1, “MyISAM Startup Options”](#).

Note that enabling external locking with --external-locking offers no protection against index corruption for tables that use delayed key writes.

- delayed_insert_limit

After inserting delayed_insert_limit delayed rows, the INSERT DELAYED handler thread checks whether there are any SELECT statements pending. If so, it allows them to execute before continuing to insert delayed rows.

- delayed_insert_timeout

How many seconds an INSERT DELAYED handler thread should wait for INSERT statements before terminating.

- delayed_queue_size

This is a per-table limit on the number of rows to queue when handling INSERT DELAYED statements. If the queue becomes full, any client that issues an INSERT DELAYED statement waits until there is room in the queue again.

- div_precision_increment

This variable indicates the number of digits of precision by which to increase the result of division operations performed with the / operator. The default value is 4. The minimum and maximum values are 0 and 30,

respectively. The following example illustrates the effect of increasing the default value.

```
mysql> SELECT 1/7;
+-----+
| 1/7    |
+-----+
| 0.1429 |
+-----+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+-----+
| 1/7                |
+-----+
| 0.142857142857    |
+-----+
```

This variable was added in MySQL 5.0.6.

- `engine_condition_pushdown`

This variable applies to NDB. By default it is 0 (OFF): If you execute a query such as `SELECT * FROM t WHERE mycol = 42`, where `mycol` is a non-indexed column, the query is executed as a full table scan on every NDB node. Each node sends every row to the MySQL server, which applies the `WHERE` condition. If `engine_condition_pushdown` is set to 1 (ON), the condition is “pushed down” to the storage engine and sent to the NDB nodes. Each node uses the condition to perform the scan, and only sends back to the MySQL server the rows that match the condition.

This variable was added in MySQL 5.0.3. Before that, the default NDB behavior is the same as for a value of OFF.

- `expire_logs_days`

The number of days for automatic binary log removal. The default is 0, which means “no automatic removal.” Possible removals happen at startup and at binary log rotation.

- `flush`

If ON, the server flushes (synchronizes) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after

each SQL statement and lets the operating system handle the synchronizing to disk. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#). This variable is set to ON if you start **mysqld** with the `--flush` option.

- `flush_time`

If this is set to a non-zero value, all tables are closed every `flush_time` seconds to free up resources and synchronize unflushed data to disk. We recommend that this option be used only on Windows 9x or Me, or on systems with minimal resources.

- `ft_boolean_syntax`

The list of operators supported by boolean full-text searches performed using `IN BOOLEAN MODE`. See [Section 12.7.1, “Boolean Full-Text Searches”](#).

The default variable value is `'+ -><()~*:""&|'`. The rules for changing the value are as follows:

- Operator function is determined by position within the string.
- The replacement value must be 14 characters.
- Each character must be an ASCII non-alphanumeric character.
- Either the first or second character must be a space.
- No duplicates are allowed except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
- Positions 10, 13, and 14 (which by default are set to `‘:’`, `‘&’`, and `‘|’`) are reserved for future extensions.

- `ft_max_word_len`

The maximum length of the word to be included in a `FULLTEXT` index.

Note: `FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_min_word_len`

The minimum length of the word to be included in a FULLTEXT index.

Note: FULLTEXT indexes must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_query_expansion_limit`

The number of top matches to use for full-text searches performed using `WITH QUERY EXPANSION`.

- `ft_stopword_file`

The file from which to read the list of stopwords for full-text searches. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the `myisam/ft_static.c` file). Setting this variable to the empty string (`' '`) disables stopwords filtering.

Note: FULLTEXT indexes must be rebuilt after changing this variable or the contents of the stopwords file. Use `REPAIR TABLE tbl_name QUICK`.

- `group_concat_max_len`

The maximum allowed result length for the `GROUP_CONCAT()` function. The default is 1024.

- `have_archive`

YES if **mysqld** supports ARCHIVE tables, NO if not.

- `have_bdb`

YES if **mysqld** supports BDB tables. DISABLED if `--skip-bdb` is used.

- `have_blackhole_engine`

YES if **mysqld** supports BLACKHOLE tables, NO if not.

- `have_compress`

YES if the `zlib` compression library is available to the server, NO if not. If not, the `COMPRESS()` and `UNCOMPRESS()` functions cannot be used.

- `have_crypt`

YES if the `crypt()` system call is available to the server, NO if not. If not, the `ENCRYPT()` function cannot be used.

- `have_csv`

YES if **mysql** supports ARCHIVE tables, NO if not.

- `have_example_engine`

YES if **mysql** supports EXAMPLE tables, NO if not.

`have_federated_engine`

YES if **mysql** supports FEDERATED tables, NO if not. This variable was added in MySQL 5.0.3.

- `have_geometry`

YES if the server supports spatial data types, NO if not.

- `have_innodb`

YES if **mysql** supports InnoDB tables. DISABLED if `--skip-innodb` is used.

- `have_isam`

In MySQL 5.0, this variable appears only for reasons of backward compatibility. It is always NO because ISAM tables are no longer supported.

- `have_ndbcluster`

YES if **mysql** supports NDB Cluster tables. DISABLED if `--skip-ndbcluster` is used.

- `have_openssl`

YES if **mysqld** supports SSL connections, NO if not.

- `have_query_cache`

YES if **mysqld** supports the query cache, NO if not.

- `have_raid`

In MySQL 5.0, this variable appears only for reasons of backward compatibility. It is always NO because RAID tables are no longer supported.

- `have_rtree_keys`

YES if RTREE indexes are available, NO if not. (These are used for spatial indexes in MyISAM tables.)

- `have_symlink`

YES if symbolic link support is enabled, NO if not. This is required on Unix for support of the `DATA DIRECTORY` and `INDEX DIRECTORY` table options, and on Windows for support of data directory symlinks.

- `init_connect`

A string to be executed by the server for each client that connects. The string consists of one or more SQL statements. To specify multiple statements, separate them by semicolon characters. For example, each client begins by default with autocommit mode enabled. There is no global system variable to specify that autocommit should be disabled by default, but `init_connect` can be used to achieve the same effect:

```
SET GLOBAL init_connect='SET AUTOCOMMIT=0';
```

This variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

```
[mysqld]  
init_connect='SET AUTOCOMMIT=0'
```

Note that the content of `init_connect` is not executed for users that have the `SUPER` privilege. This is done so that an erroneous value for

`init_connect` does not prevent all clients from connecting. For example, the value might contain a statement that has a syntax error, thus causing client connections to fail. Not executing `init_connect` for users that have the `SUPER` privilege enables them to open a connection and fix the `init_connect` value.

- `init_file`

The name of the file specified with the `--init-file` option when you start the server. This should be a file containing SQL statements that you want the server to execute when it starts. Each statement must be on a single line and should not include comments.

- `init_slave`

This variable is similar to `init_connect`, but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` variable.

- `innodb_xxx`

InnoDB system variables are listed in [Section 14.2.4, “InnoDB Startup Options and System Variables”](#).

- `interactive_timeout`

The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the `CLIENT_INTERACTIVE` option to `mysql_real_connect()`. See also `wait_timeout`.

- `join_buffer_size`

The size of the buffer that is used for joins that do not use indexes and thus perform full table scans. Normally, the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary.

- `key_buffer_size`

Index blocks for MyISAM tables are buffered and are shared by all threads. `key_buffer_size` is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

The maximum allowable setting for `key_buffer_size` is 4GB. The effective maximum size might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware platform.

Increase the value to get better index handling (for all reads and multiple writes) to as much as you can afford. Using a value that is 25% of total memory on a machine that mainly runs MySQL is quite common. However, if you make the value too large (for example, more than 50% of your total memory) your system might start to page and become extremely slow. MySQL relies on the operating system to perform filesystem caching for data reads, so you must leave some room for the filesystem cache. Consider also the memory requirements of other storage engines.

For even more speed when writing many rows at the same time, use `LOCK TABLES`. See [Section 7.2.16, “Speed of INSERT Statements”](#).

You can check the performance of the key buffer by issuing a `SHOW STATUS` statement and examining the `Key_read_requests`, `Key_reads`, `Key_write_requests`, and `Key_writes` status variables. (See [Section 13.5.4, “SHOW Syntax”](#).) The `Key_reads/Key_read_requests` ratio should normally be less than 0.01. The `Key_writes/Key_write_requests` ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the `DELAY_KEY_WRITE` table option.

The fraction of the key buffer in use can be determined using `key_buffer_size` in conjunction with the `Key_blocks_unused` status variable and the buffer block size, which is available from the `key_cache_block_size` system variable:

$$1 - ((\text{Key_blocks_unused} \times \text{key_cache_block_size}) / \text{key_buffer_siz})$$

This value is an approximation because some space in the key buffer may

be allocated internally for administrative structures.

It is possible to create multiple MyISAM key caches. The size limit of 4GB applies to each cache individually, not as a group. See [Section 7.4.6, “The MyISAM Key Cache”](#).

- `key_cache_age_threshold`

This value controls the demotion of buffers from the hot sub-chain of a key cache to the warm sub-chain. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. See [Section 7.4.6, “The MyISAM Key Cache”](#).

- `key_cache_block_size`

The size in bytes of blocks in the key cache. The default value is 1024. See [Section 7.4.6, “The MyISAM Key Cache”](#).

- `key_cache_division_limit`

The division point between the hot and warm sub-chains of the key cache buffer chain. The value is the percentage of the buffer chain to use for the warm sub-chain. Allowable values range from 1 to 100. The default value is 100. See [Section 7.4.6, “The MyISAM Key Cache”](#).

- `language`

The language used for error messages.

- `large_file_support`

Whether **mysqld** was compiled with options for large file support.

- `large_pages`

Whether large page support is enabled. This variable was added in MySQL 5.0.3.

- `license`

The type of license the server has.

- `local_infile`

Whether LOCAL is supported for LOAD DATA INFILE statements. See [Section 5.7.4, “Security Issues with LOAD DATA LOCAL”](#).

- `locked_in_memory`

Whether **mysqld** was locked in memory with `--memlock`.

- `log`

Whether logging of all statements to the general query log is enabled. See [Section 5.12.2, “The General Query Log”](#).

- `log_bin`

Whether the binary log is enabled. See [Section 5.12.3, “The Binary Log”](#).

- `log_bin_trust_function_creators`

This variable applies when binary logging is enabled. It controls whether stored function creators can be trusted not to create stored functions that will cause unsafe events to be written to the binary log. If set to 0 (the default), users are not allowed to create or alter stored functions unless they have the SUPER privilege in addition to the CREATE ROUTINE or ALTER ROUTINE privilege. A setting of 0 also enforces the restriction that a function must be declared with the DETERMINISTIC characteristic, or with the READS SQL DATA or NO SQL characteristic. If the variable is set to 1, MySQL does not enforce these restrictions on stored function creation. See [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

This variable was added in MySQL 5.0.16.

- `log_bin_trust_routine_creators`

This is the old name for `log_bin_trust_function_creators`. Before MySQL 5.0.16, it also applies to stored procedures, not just stored functions. As of 5.0.16, this variable is deprecated. It is recognized for backward compatibility but its use results in a warning.

This variable was added in MySQL 5.0.6.

- `log_error`

The location of the error log.

- `log_queries_not_using_indexes`

Whether queries that do not use indexes are logged to the slow query log. See [Section 5.12.4, “The Slow Query Log”](#). This variable was added in MySQL 5.0.23.

- `log_slave_updates`

Whether updates received by a slave server from a master server should be logged to the slave's own binary log. Binary logging must be enabled on the slave for this variable to have any effect. See [Section 6.8, “Replication Startup Options”](#).

- `log_slow_queries`

Whether slow queries should be logged. “Slow” is determined by the value of the `long_query_time` variable. See [Section 5.12.4, “The Slow Query Log”](#).

- `log_warnings`

Whether to produce additional warning messages. It is enabled (1) by default. Aborted connections are not logged to the error log unless the value is greater than 1.

- `long_query_time`

If a query takes longer than this many seconds, the server increments the `Slow_queries` status variable. If you are using the `--log-slow-queries` option, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum value is 1. The default is 10. See [Section 5.12.4, “The Slow Query Log”](#).

- `low_priority_updates`

If set to 1, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. This variable previously was named `sql_low_priority_updates`.

- `lower_case_file_system`

This variable describes the case sensitivity of filenames on the filesystem where the data directory is located. `OFF` means filenames are case sensitive, `ON` means they are not case sensitive.

- `lower_case_table_names`

If set to 1, table names are stored in lowercase on disk and table name comparisons are not case sensitive. If set to 2 table names are stored as given but compared in lowercase. This option also applies to database names and table aliases. See [Section 9.2.2, “Identifier Case Sensitivity”](#).

If you are using InnoDB tables, you should set this variable to 1 on all platforms to force names to be converted to lowercase.

You should *not* set this variable to 0 if you are running MySQL on a system that does not have case-sensitive filenames (such as Windows or Mac OS X). If this variable is not set at startup and the filesystem on which the data directory is located does not have case-sensitive filenames, MySQL automatically sets `lower_case_table_names` to 2.

- `max_allowed_packet`

The maximum size of one packet or any generated/intermediate string.

The packet message buffer is initialized to `net_buffer_length` bytes, but can grow up to `max_allowed_packet` bytes when needed. This value by default is small, to catch large (possibly incorrect) packets.

You must increase this value if you are using large `BLOB` columns or long strings. It should be as big as the largest `BLOB` you want to use. The protocol limit for `max_allowed_packet` is 1GB.

- `max_binlog_cache_size`

If a multiple-statement transaction requires more than this amount of memory, the server generates a Multi-statement transaction required more than '`max_binlog_cache_size`' bytes of storage error.

- `max_binlog_size`

If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). You cannot set this variable to more than 1GB or to less than 4096 bytes. The default value is 1GB.

A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary logs larger than `max_binlog_size`.

If `max_relay_log_size` is 0, the value of `max_binlog_size` applies to relay logs as well.

- `max_connect_errors`

If there are more than this number of interrupted connections from a host, that host is blocked from further connections. You can unblock blocked hosts with the `FLUSH HOSTS` statement.

- `max_connections`

The number of simultaneous client connections allowed. Increasing this value increases the number of file descriptors that **mysqld** requires. See [Section 7.4.8, “How MySQL Opens and Closes Tables”](#), for comments on file descriptor limits. See also [Section A.2.6, “Too many connections”](#).

- `max_delayed_threads`

Do not start more than this number of threads to handle `INSERT DELAYED` statements. If you try to insert data into a new table after all `INSERT DELAYED` threads are in use, the row is inserted as if the `DELAYED` attribute wasn't specified. If you set this to 0, MySQL never creates a thread to handle `DELAYED` rows; in effect, this disables `DELAYED` entirely.

- `max_error_count`

The maximum number of error, warning, and note messages to be stored for display by the `SHOW ERRORS` and `SHOW WARNINGS` statements.

- `max_heap_table_size`

This variable sets the maximum size to which `MEMORY` tables are allowed to grow. The value of the variable is used to calculate `MEMORY` table `MAX_ROWS` values. Setting this variable has no effect on any existing `MEMORY` table, unless the table is re-created with a statement such as `CREATE TABLE` or altered with `ALTER TABLE` or `TRUNCATE TABLE`.

- `max_insert_delayed_threads`

This variable is a synonym for `max_delayed_threads`.

- `max_join_size`

Do not allow `SELECT` statements that probably need to examine more than `max_join_size` rows (for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than `max_join_size` disk seeks. By setting this value, you can catch `SELECT` statements where keys are not used properly and that would probably take a long time. Set it if your users tend to perform joins that lack a `WHERE` clause, that take a long time, or that return millions of rows.

Setting this variable to a value other than `DEFAULT` resets the value of `SQL_BIG_SELECTS` to 0. If you set the `SQL_BIG_SELECTS` value again, the `max_join_size` variable is ignored.

If a query result is in the query cache, no result size check is performed, because the result has previously been computed and it does not burden the server to send it to the client.

This variable previously was named `sql_max_join_size`.

- `max_length_for_sort_data`

The cutoff on the size of index values that determines which `filesort`

algorithm to use. See [Section 7.2.12, “ORDER BY Optimization”](#).

- `max_prepared_stmt_count`

This variable limits the total number of prepared statements in the server. It can be used in environments where there is the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. The default value is 16,382. The allowable range of values is from 0 to 1 million. If the value is set lower than the current number of prepared statements, existing statements are not affected and can be used, but no new statements can be prepared until the current number drops below the limit. This variable was added in MySQL 5.0.21.

- `max_relay_log_size`

If a write by a replication slave to its relay log causes the current log file size to exceed the value of this variable, the slave rotates the relay logs (closes the current file and opens the next one). If `max_relay_log_size` is 0, the server uses `max_binlog_size` for both the binary log and the relay log. If `max_relay_log_size` is greater than 0, it constrains the size of the relay log, which enables you to have different sizes for the two logs. You must set `max_relay_log_size` to between 4096 bytes and 1GB (inclusive), or to 0. The default value is 0. See [Section 6.3, “Replication Implementation Details”](#).

- `max_seeks_for_key`

Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index (see [Section 13.5.4.13, “SHOW INDEX Syntax”](#)). By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

- `max_sort_length`

The number of bytes to use when sorting BLOB or TEXT values. Only the first `max_sort_length` bytes of each value are used; the rest are ignored.

- `max_sp_recursion_depth`

The number of times that a stored procedure may call itself. The default value for this option is 0, which completely disallows recursion in stored procedures. The maximum value is 255.

This variable can be set globally and per session.

- `max_tmp_tables`

The maximum number of temporary tables a client can keep open at the same time. (This option does not yet do anything.)

- `max_user_connections`

The maximum number of simultaneous connections allowed to any given MySQL account. A value of 0 means “no limit.”

Before MySQL 5.0.3, this variable has only global scope. Beginning with MySQL 5.0.3, it also has a read-only session scope. The session variable has the same value as the global variable unless the current account has a non-zero `MAX_USER_CONNECTIONS` resource limit. In that case, the session value reflects the account limit.

- `max_write_lock_count`

After this many write locks, allow some pending read lock requests to be processed in between.

- `myisam_data_pointer_size`

The default pointer size in bytes, to be used by `CREATE TABLE` for MyISAM tables when no `MAX_ROWS` option is specified. This variable cannot be less than 2 or larger than 7. The default value is 6 (4 before MySQL 5.0.6). This variable was added in MySQL 4.1.2. See [Section A.2.11, “The table is full”](#).

- `myisam_max_extra_sort_file_size` (*DEPRECATED*)

If the temporary file used for fast MyISAM index creation would be larger than using the key cache by the amount specified here, prefer the key cache method. This is mainly used to force long character keys in large tables to

use the slower key cache method to create the index. The value is given in bytes.

Note: This variable was removed in MySQL 5.0.6.

- `myisam_max_sort_file_size`

The maximum size of the temporary file that MySQL is allowed to use while re-creating a MyISAM index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

- `myisam_recover_options`

The value of the `--myisam-recover` option. See [Section 5.2.1, “mysqld Command Options”](#).

- `myisam_repair_threads`

If this value is greater than 1, MyISAM table indexes are created in parallel (each index in its own thread) during the Repair by sorting process. The default value is 1.

Note: Multi-threaded repair is still *beta-quality* code.

- `myisam_sort_buffer_size`

The size of the buffer that is allocated when sorting MyISAM indexes during a `REPAIR TABLE` or when creating indexes with `CREATE INDEX` or `ALTER TABLE`.

- `myisam_stats_method`

How the server treats `NULL` values when collecting statistics about the distribution of index values for MyISAM tables. This variable has two possible values, `nulls_equal` and `nulls_unequal`. For `nulls_equal`, all `NULL` index values are considered equal and form a single value group that has a size equal to the number of `NULL` values. For `nulls_unequal`, `NULL` values are considered unequal, and each `NULL` forms a distinct value group

of size 1.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in [Section 7.4.7, “MyISAM Index Statistics Collection”](#).

This variable was added in MySQL 5.0.14. For older versions, the statistics collection method is equivalent to `nulls_equal`.

- `multi_read_range`

Specifies the maximum number of ranges to send to a storage engine during range selects. The default value is 256. Sending multiple ranges to an engine is a feature that can improve the performance of certain selects dramatically, particularly for `NDBCLUSTER`. This engine needs to send the range requests to all nodes, and sending many of those requests at once reduces the communication costs significantly. This variable was added in MySQL 5.0.3.

- `named_pipe`

(Windows only.) Indicates whether the server supports connections over named pipes.

- `ndb_autoincrement_prefetch_sz`

Determines the probability of gaps in an autoincremented column. Set to 1 to minimize this. Set to a high value for optimization — makes inserts faster, but decreases the likelihood that consecutive autoincrement numbers will be used in a batch of inserts. Default value: 32. Minimum value: 1.

- `ndb_cache_check_time`

The number of milliseconds to wait before checking the NDB query cache. Setting this to 0 (the default and minimum value) means that the NDB query cache will be checked for validation on every query.

The recommended maximum value for this variable is 1000, which means that the query cache is checked once per second. A larger value means the NDB query cache is less often checked and invalidated due to updates on a

different **mysqld**. It is generally not desirable to set this to a value greater than 2000.

- `ndb_force_send`

Forces sending of buffers to NDB immediately, without waiting for other threads. Defaults to ON.

- `ndb_index_stat_cache_entries`

Sets the granularity of the statistics by determining the number of starting and ending keys to store in the statistics memory cache. Zero means no caching takes place; in this case, the data nodes are always queries directly. Default value: 32.

- `ndb_index_stat_enable`

Use NDB index statistics in query optimization. Defaults to ON.

- `ndb_index_stat_update_freq`

How often to query data nodes instead of the statistics cache. For example, a value of 20 (the default) means to direct every 20th query to the data nodes.

- `ndb_report_thresh_binlog_epoch_slip`

This is a threshold on the number of epochs to be behind before reporting binlog status. For example, a value of 3 (the default) means that if the difference between which epoch has been received from the storage nodes and which epoch has been applied to the binlog is 3 or more, a status message will be sent to the cluster log.

- `ndb_report_thresh_binlog_mem_usage`

This is a threshold on the percentage of free memory remaining before reporting binlog status. For example, a value of 10 (the default) means that if the amount of available memory for receiving binlog data from the data nodes falls below 10%, a status message will be sent to the cluster log.

- `ndb_use_exact_count`

Forces NDB to use an count of records during `SELECT COUNT(*)` query planning to speed up this type of query. The default value is `ON`. For faster queries overall, disable this feature by setting the value of `ndb_use_exact_count` to `OFF`.

- `ndb_use_transactions`

You can disable NDB transaction support by setting this variable's values to `OFF` (not recommended). The default is `ON`.

- `net_buffer_length`

The communication buffer is reset to this size between SQL statements. This variable should not normally be changed, but if you have very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the buffer is automatically enlarged, up to `max_allowed_packet` bytes.

- `net_read_timeout`

The number of seconds to wait for more data from a connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made via Unix socket files, named pipes, or shared memory. When the server is reading from the client, `net_read_timeout` is the timeout value controlling when to abort. When the server is writing to the client, `net_write_timeout` is the timeout value controlling when to abort. See also `slave_net_timeout`.

- `net_retry_count`

If a read on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads.

- `net_write_timeout`

The number of seconds to wait for a block to be written to a connection before aborting the write. This timeout applies only to TCP/IP connections,

not to connections made via Unix socket files, named pipes, or shared memory. See also `net_read_timeout`.

- `new`

This variable was used in MySQL 4.0 to turn on some 4.1 behaviors, and is retained for backward compatibility. In MySQL 5.0, its value is always `OFF`.

- `old_passwords`

Whether the server should use pre-4.1-style passwords for MySQL user accounts. See [Section A.2.3, “Client does not support authentication protocol”](#).

- `one_shot`

This is not a variable, but it can be used when setting some variables. It is described in [Section 13.5.3, “SET Syntax”](#).

- `open_files_limit`

The number of files that the operating system allows **mysqld** to open. This is the real value allowed by the system and might be different from the value you gave using the `--open-files-limit` option to **mysqld** or **mysqld_safe**. The value is 0 on systems where MySQL can't change the number of open files.

- `optimizer_prune_level`

Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space. A value of 0 disables heuristics so that the optimizer performs an exhaustive search. A value of 1 causes the optimizer to prune plans based on the number of rows retrieved by intermediate plans. This variable was added in MySQL 5.0.1.

- `optimizer_search_depth`

The maximum depth of search performed by the query optimizer. Values larger than the number of relations in a query result in better query plans, but take longer to generate an execution plan for a query. Values smaller

than the number of relations in a query return an execution plan quicker, but the resulting plan may be far from being optimal. If set to 0, the system automatically picks a reasonable value. If set to the maximum number of tables used in a query plus 2, the optimizer switches to the algorithm used in MySQL 5.0.0 (and previous versions) for performing searches. This variable was added in MySQL 5.0.1.

- `pid_file`

The pathname of the process ID (PID) file. This variable can be set with the `--pid-file` option.

- `port`

The number of the port on which the server listens for TCP/IP connections. This variable can be set with the `--port` option.

- `preload_buffer_size`

The size of the buffer that is allocated when preloading indexes.

- `prepared_stmt_count`

The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` system variable.) This variable was added in MySQL 5.0.21.

- `protocol_version`

The version of the client/server protocol used by the MySQL server.

- `query_alloc_block_size`

The allocation size of memory blocks that are allocated for objects created during statement parsing and execution. If you have problems with memory fragmentation, it might help to increase this a bit.

- `query_cache_limit`

Don't cache results that are larger than this number of bytes. The default

value is 1MB.

- `query_cache_min_res_unit`

The minimum size (in bytes) for blocks allocated by the query cache. The default value is 4096 (4KB). Tuning information for this variable is given in [Section 5.14.3, “Query Cache Configuration”](#).

- `query_cache_size`

The amount of memory allocated for caching query results. The default value is 0, which disables the query cache. The allowable values are multiples of 1024; other values are rounded down to the nearest multiple. Note that `query_cache_size` bytes of memory are allocated even if `query_cache_type` is set to 0. See [Section 5.14.3, “Query Cache Configuration”](#), for more information.

- `query_cache_type`

Set the query cache type. Setting the GLOBAL value sets the type for all clients that connect thereafter. Individual clients can set the SESSION value to affect their own use of the query cache. Possible values are shown in the following table:

Option	Description
0 or OFF	Don't cache results in or retrieve results from the query cache. Note that this does not deallocate the query cache buffer. To do that, you should set <code>query_cache_size</code> to 0.
1 or ON	Cache all query results except for those that begin with <code>SELECT SQL_NO_CACHE</code> .
2 or DEMAND	Cache results only for queries that begin with <code>SELECT SQL_CACHE</code> .

This variable defaults to ON.

- `query_cache_wlock_invalidate`

Normally, when one client acquires a WRITE lock on a MyISAM table, other clients are not blocked from issuing statements that read from the table if

the query results are present in the query cache. Setting this variable to 1 causes acquisition of a `WRITE` lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect.

- `query_prealloc_size`

The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger `query_prealloc_size` value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

- `range_alloc_block_size`

The size of blocks that are allocated when doing range optimization.

- `read_buffer_size`

Each thread that does a sequential scan allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you might want to increase this value, which defaults to 131072.

- `read_only`

When the variable is set to `ON` for a replication slave server, it causes the slave to allow no updates except from slave threads or from users that have the `SUPER` privilege. This can be useful to ensure that a slave server accepts updates only from its master server and not from clients. As of MySQL 5.0.16, this variable does not apply to `TEMPORARY` tables.

- `relay_log_purge`

Disables or enables automatic purging of relay log files as soon as they are not needed any more. The default value is 1 (`ON`).

- `read_rnd_buffer_size`

When reading rows in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks. Setting the variable to

a large value can improve ORDER BY performance by a lot. However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries.

- `rpl_recovery_rank`

This variable is unused.

- `secure_auth`

If the MySQL server has been started with the `--secure-auth` option, it blocks connections from all accounts that have passwords stored in the old (pre-4.1) format. In that case, the value of this variable is `ON`, otherwise it is `OFF`.

You should enable this option if you want to prevent all use of passwords employing the old format (and hence insecure communication over the network).

Server startup fails with an error if this option is enabled and the privilege tables are in pre-4.1 format. See [Section A.2.3, “Client does not support authentication protocol”](#).

- `server_id`

The server ID. This value is set by the `--server-id` option. It is used for replication to enable master and slave servers to identify themselves uniquely.

- `shared_memory`

(Windows only.) Whether the server allows shared-memory connections.

- `shared_memory_base_name`

(Windows only.) The name of shared memory to use for shared-memory connections. This is useful when running multiple MySQL instances on a single physical machine. The default name is `MYSQL`. The name is case sensitive.

- skip_external_locking

This is OFF if **mysqld** uses external locking, ON if external locking is disabled.

- skip_networking

This is ON if the server allows only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe or shared memory. On NetWare, only TCP/IP connections are supported, so do not set this variable to ON. This variable can be set to ON with the --skip-networking option.

- skip_show_database

This prevents people from using the SHOW DATABASES statement if they do not have the SHOW DATABASES privilege. This can improve security if you have concerns about users being able to see databases belonging to other users. Its effect depends on the SHOW DATABASES privilege: If the variable value is ON, the SHOW DATABASES statement is allowed only to users who have the SHOW DATABASES privilege, and the statement displays all database names. If the value is OFF, SHOW DATABASES is allowed to all users, but displays the names of only those databases for which the user has the SHOW DATABASES or other privilege.

- slave_compressed_protocol

Whether to use compression of the slave/master protocol if both the slave and the master support it.

- slave_load_tmpdir

The name of the directory where the slave creates temporary files for replicating LOAD DATA INFILE statements.

- slave_net_timeout

The number of seconds to wait for more data from a master/slave connection before aborting the read. This timeout applies only to TCP/IP connections, not to connections made via Unix socket files, named pipes, or

shared memory.

- `slave_skip_errors`

The replication errors that the slave should skip (ignore).

- `slave_transaction_retries`

If a replication slave SQL thread fails to execute a transaction because of an InnoDB deadlock or exceeded InnoDB's `innodb_lock_wait_timeout` or NDBCluster's `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout`, it automatically retries `slave_transaction_retries` times before stopping with an error. The default prior to MySQL 4.0.3 is 0. You must explicitly set the value greater than 0 to enable the “retry” behavior, which is probably a good idea. In MySQL 5.0.3 or newer, the default is 10.

- `slow_launch_time`

If creating a thread takes longer than this many seconds, the server increments the `Slow_launch_threads` status variable.

- `socket`

On Unix platforms, this variable is the name of the socket file that is used for local client connections. The default is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On Windows, this variable is the name of the named pipe that is used for local client connections. The default value is `MySQL` (not case sensitive).

- `sort_buffer_size`

Each thread that needs to do a sort allocates a buffer of this size. Increase this value for faster `ORDER BY` or `GROUP BY` operations. See [Section A.4.4, “Where MySQL Stores Temporary Files”](#).

- `sql_mode`

The current server SQL mode, which can be set dynamically. See [Section 5.2.5, “The Server SQL Mode”](#).

- `sql_slave_skip_counter`

The number of events from the master that a slave server should skip. See [Section 13.6.2.6, “SET GLOBAL SQL_SLAVE_SKIP_COUNTER Syntax”](#).

- `ssl_ca`

The path to a file with a list of trusted SSL CAs. This variable was added in MySQL 5.0.23.

- `ssl_capath`

The path to a directory that contains trusted SSL CA certificates in PEM format. This variable was added in MySQL 5.0.23.

- `ssl_cert`

The name of the SSL certificate file to use for establishing a secure connection. This variable was added in MySQL 5.0.23.

- `ssl_cipher`

A list of allowable ciphers to use for SSL encryption. The cipher list has the same format as the `openssl ciphers` command. This variable was added in MySQL 5.0.23.

- `ssl_key`

The name of the SSL key file to use for establishing a secure connection. This variable was added in MySQL 5.0.23.

- `storage_engine`

The default storage engine (table type). To set the storage engine at server startup, use the `--default-storage-engine` option. See [Section 5.2.1, “`mysqld` Command Options”](#).

- `sync_binlog`

If the value of this variable is positive, the MySQL server synchronizes its binary log to disk (using `fdatasync()`) after every `sync_binlog` writes to the binary log. Note that there is one write to the binary log per statement if autocommit is enabled, and one write per transaction otherwise. The default value is 0, which does no synchronizing to disk. A value of 1 is the safest choice, because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

If the value of `sync_binlog` is 0 (the default), no extra flushing is done. The server relies on the operating system to flush the file contents occasionally as for any other file.

- `sync_frm`

If this variable is set to 1, when any non-temporary table is created its `.frm` file is synchronized to disk (using `fdatasync()`). This is slower but safer in case of a crash. The default is 1.

- `system_time_zone`

The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set `system_time_zone`. Typically the time zone is specified by the `TZ` environment variable. It also can be specified using the `--timezone` option of the **mysqld_safe** script.

The `system_time_zone` variable differs from `time_zone`. Although they might have the same value, the latter variable is used to initialize the time zone for each client that connects. See [Section 5.11.8, “MySQL Server Time Zone Support”](#).

- `table_cache`

The number of open tables for all threads. Increasing this value increases the number of file descriptors that **mysqld** requires. You can check whether you need to increase the table cache by checking the `Opened_tables` status variable. See [Section 5.2.4, “Server Status Variables”](#). If the value of

Opened_tables is large and you don't do FLUSH TABLES often (which just forces all tables to be closed and reopened), then you should increase the value of the table_cache variable. For more information about the table cache, see [Section 7.4.8, “How MySQL Opens and Closes Tables”](#).

- table_lock_wait_timeout

Specifies a wait timeout for table-level locks, in seconds. The default timeout is 50 seconds. The timeout is active only if the connection has open cursors. This variable can also be set globally at runtime (you need the SUPER privilege to do this). It's available as of MySQL 5.0.10.

- table_type

This variable is a synonym for storage_engine. In MySQL 5.0, storage_engine is the preferred name.

- thread_cache_size

How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than thread_cache_size threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. (Normally, this doesn't provide a notable performance improvement if you have a good thread implementation.) By examining the difference between the Connections and Threads_created status variables, you can see how efficient the thread cache is. For details, see [Section 5.2.4, “Server Status Variables”](#).

- thread_concurrency

On Solaris, **mysqld** calls thr_setconcurrency() with this value. This function enables applications to give the threads system a hint about the desired number of threads that should be run at the same time.

- thread_stack

The stack size for each thread. Many of the limits detected by the crash-me test are dependent on this value. The default is large enough for normal

operation. See [Section 7.1.4, “The MySQL Benchmark Suite”](#). The default is 192KB.

- `time_format`

This variable is not implemented.

- `time_zone`

The current time zone. This variable is used to initialize the time zone for each client that connects. By default, the initial value of this is 'SYSTEM' (which means, “use the value of `system_time_zone`”). The value can be specified explicitly at server startup with the `--default-time-zone` option. See [Section 5.11.8, “MySQL Server Time Zone Support”](#).

- `tmp_table_size`

The maximum size of in-memory temporary tables. (The actual limit is determined as the smaller of `max_heap_table_size` and `tmp_table_size`.) If an in-memory temporary table exceeds the limit, MySQL automatically converts it to an on-disk MyISAM table. Increase the value of `tmp_table_size` (and `max_heap_table_size` if necessary) if you do many advanced GROUP BY queries and you have lots of memory.

- `tmpdir`

The directory used for temporary files and temporary tables. This variable can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (':') on Unix and semicolon characters (';') on Windows, NetWare, and OS/2.

The multiple-directory feature can be used to spread the load between several physical disks. If the MySQL server is acting as a replication slave, you should not set `tmpdir` to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails. However, if you are using MySQL 4.0.0 or later, you can set the slave's temporary directory using the `slave_load_tmpdir`

variable. In that case, the slave won't use the general `tmpdir` value and you can set `tmpdir` to a non-permanent location.

- `transaction_alloc_block_size`

The amount in bytes by which to increase a per-transaction memory pool which needs memory. See the description of `transaction_prealloc_size`.

- `transaction_prealloc_size`

There is a per-transaction memory pool from which various transaction-related allocations take memory. The initial size of the pool in bytes is `transaction_prealloc_size`. For every allocation that cannot be satisfied from the pool because it has insufficient memory available, the pool is increased by `transaction_alloc_block_size` bytes. When the transaction ends, the pool is truncated to `transaction_prealloc_size` bytes.

By making `transaction_prealloc_size` sufficiently large to contain all statements within a single transaction, you can avoid many `malloc()` calls.

- `tx_isolation`

The default transaction isolation level. Defaults to `REPEATABLE-READ`.

This variable is set by the `SET TRANSACTION ISOLATION LEVEL` statement. See [Section 13.4.6, “SET TRANSACTION Syntax”](#). If you set `tx_isolation` directly to an isolation level name that contains a space, the name should be enclosed within quotes, with the space replaced by a dash. For example:

```
SET tx_isolation = 'READ-COMMITTED';
```

- `updatable_views_with_limit`

This variable controls whether updates to a view can be made when the view does not contain all columns of the primary key defined in the underlying table, if the update statement contains a `LIMIT` clause. (Such updates often are generated by GUI tools.) An update is an `UPDATE` or `DELETE` statement. Primary key here means a `PRIMARY KEY`, or a `UNIQUE` index in which no column can contain `NULL`.

The variable can have two values:

- 1 or YES: Issue a warning only (not an error message). This is the default value.
- 0 or NO: Prohibit the update.

This variable was added in MySQL 5.0.2.

- `version`

The version number for the server.

- `version_bdb`

The BDB storage engine version.

- `version_comment`

The **configure** script has a `--with-comment` option that allows a comment to be specified when building MySQL. This variable contains the value of that comment.

- `version_compile_machine`

The type of machine or architecture on which MySQL was built.

- `version_compile_os`

The type of operating system on which MySQL was built.

- `wait_timeout`

The number of seconds the server waits for activity on a non-interactive connection before closing it. This timeout applies only to TCP/IP connections, not to connections made via Unix socket files, named pipes, or shared memory.

On thread startup, the session `wait_timeout` value is initialized from the global `wait_timeout` value or from the global `interactive_timeout` value, depending on the type of client (as defined by the `CLIENT_INTERACTIVE`

connect option to `mysql_real_connect()`). See also `interactive_timeout`.

5.2.3. Using System Variables

The **mysql** server maintains many system variables that indicate how it is configured. [Section 5.2.2, “Server System Variables”](#), describes the meaning of these variables. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

The server maintains two kinds of system variables. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections. A given system variable can have both a global and a session value. Global and session system variables are related as follows:

- When the server starts, it initializes all global variables to their default values. These defaults can be changed by options specified on the command line or in an option file. (See [Section 4.3, “Specifying Program Options”](#).)
- The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For example, the client's SQL mode is controlled by the session `sql_mode` value, which is initialized when the client connects to the value of the global `sql_mode` value.

System variable values can be set globally at server startup by using options on the command line or in an option file. When you use a startup option to set a variable that takes a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024 , 1024^2 or 1024^3 ; that is, units of kilobytes, megabytes, or gigabytes, respectively. Thus, the following command starts the server with a query cache size of 16 megabytes and a maximum packet size of one gigabyte:

```
mysqld --query_cache_size=16M --max_allowed_packet=1G
```

Within an option file, those variables are set like this:

```
[mysqld]
query_cache_size=16M
max_allowed_packet=1G
```

The lettercase of suffix letters does not matter; 16M and 16m are equivalent, as are 1G and 1g.

If you want to restrict the maximum value to which a system variable can be set at runtime with the SET statement, you can specify this maximum by using an option of the form `--maximum-var_name=value` at server startup. For example, to prevent the value of `query_cache_size` from being increased to more than 32MB at runtime, use the option `--maximum-query_cache_size=32M`.

Many system variables are dynamic and can be changed while the server runs by using the SET statement. For a list, see [Section 5.2.3.2, “Dynamic System Variables”](#). To change a system variable with SET, refer to it as `var_name`, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by GLOBAL or @@global.. The SUPER privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by SESSION, @@session., or @@. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- LOCAL and @@local. are synonyms for SESSION and @@session..
- If no modifier is present, SET changes the session variable.

A SET statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent GLOBAL or SESSION modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
```

```
SET @@local.sort_buffer_size=10000;  
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=100000  
SET @@sort_buffer_size=1000000;  
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000
```

When you assign a value to a system variable with SET, you cannot use suffix letters in the value (as can be done with startup options). However, the value can take the form of an expression:

```
SET sort_buffer_size = 10 * 1024 * 1024;
```

The @@var_name syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the SET GLOBAL statement).

To prevent incorrect usage, MySQL produces an error if you use SET GLOBAL with a variable that can only be used with SET SESSION or if you do not specify GLOBAL (or @@global.) when setting a global variable.

To set a SESSION variable to the GLOBAL value or a GLOBAL value to the compiled-in MySQL default value, use the DEFAULT keyword. For example, the following two statements are identical in setting the session value of max_join_size to the global value:

```
SET max_join_size=DEFAULT;  
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to DEFAULT. In such cases, use of DEFAULT results in an error.

You can refer to the values of specific global or session system variables in expressions by using one of the @@-modifiers. For example, you can retrieve values in a SELECT statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as @@var_name (that is, when you do not specify @@global. or @@session.), MySQL returns the session value if it exists and the global value otherwise. (This differs from SET @@var_name = value, which always refers to the session value.)

Note: Some system variables can be enabled with the SET statement by setting them to ON or 1, or disabled by setting them to OFF or 0. However, to set such a variable on the command line or in an option file, you must set it to 1 or 0; setting it to ON or OFF will not work. For example, on the command line, --delay_key_write=1 works but --delay_key_write=ON does not.

To display system variable names and values, use the SHOW VARIABLES statement.

```
mysql> SHOW VARIABLES;
```

Variable_name	Value
auto_increment_increment	1
auto_increment_offset	1
automatic_sp_privileges	ON
back_log	50
basedir	/
bdb_cache_size	8388600
bdb_home	/var/lib/mysql/
bdb_log_buffer_size	32768
bdb_logdir	
bdb_max_lock	10000
bdb_shared_data	OFF
bdb_tmpdir	/tmp/
binlog_cache_size	32768
bulk_insert_buffer_size	8388608
character_set_client	latin1
character_set_connection	latin1
character_set_database	latin1
character_set_results	latin1
character_set_server	latin1
character_set_system	utf8
character_sets_dir	/usr/share/mysql/charsets/

```

| collation_connection          | latin1_swedish_ci
| collation_database           | latin1_swedish_ci
| collation_server              | latin1_swedish_ci
...
| innodb_additional_mem_pool_size | 1048576
| innodb_autoextend_increment    | 8
| innodb_buffer_pool_ave_mem_mb  | 0
| innodb_buffer_pool_size       | 8388608
| innodb_checksums              | ON
| innodb_commit_concurrency      | 0
| innodb_concurrency_tickets     | 500
| innodb_data_file_path          | ibdata1:10M:autoextend
| innodb_data_home_dir           |
...
| version                       | 5.0.19-Max
| version_comment                | MySQL Community Edition - Max (G
| version_compile_machine        | i686
| version_compile_os             | pc-linux-gnu
| wait_timeout                   | 28800
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

With a LIKE clause, the statement displays only those variables that match the pattern. To obtain a specific variable name, use a LIKE clause as shown:

```

SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';

```

To get a list of variables whose name match a pattern, use the ‘%’ wildcard character in a LIKE clause:

```

SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';

```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because ‘_’ is a wildcard that matches any single character, you should escape it as ‘_’ to match it literally. In practice, this is rarely necessary.

For SHOW VARIABLES, if you specify neither GLOBAL nor SESSION, MySQL returns SESSION values.

The reason for requiring the GLOBAL keyword when setting GLOBAL-only variables but not when retrieving them is to prevent problems in the future. If we were to remove a SESSION variable that has the same name as a GLOBAL variable, a client with the SUPER privilege might accidentally change the GLOBAL variable

rather than just the `SESSION` variable for its own connection. If we add a `SESSION` variable with the same name as a `GLOBAL` variable, a client that intends to change the `GLOBAL` variable might find only its own `SESSION` variable changed.

5.2.3.1. Structured System Variables

A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.
- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

MySQL 5.0 supports one structured variable type, which specifies parameters governing the operation of key caches. A key cache structured variable has these components:

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

This section describes the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in [Section 7.4.6, “The MyISAM Key Cache”](#).

To refer to a component of a structured variable instance, you can use a compound name in `instance_name.component_name` format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of `default` is always predefined. If you refer to a component of a structured variable without

any instance name, the default instance is used. Thus, `default.key_buffer_size` and `key_buffer_size` both refer to the same system variable.

Structured variable instances and components follow these naming rules:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named `default`, so `default` is not unique across variable types.
- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.
- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, `hot-cache` is not legal, but ``hot-cache`` is.
- `global`, `session`, and `local` are not legal instance names. This avoids a conflict with notation such as `@@global.var_name` for referring to non-structured system variables.

Currently, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules will assume greater significance if some other type of structured variable is created in the future.

With one exception, you can refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

In an option file, use this syntax:

```
[mysqld]  
hot_cache.key_buffer_size=64K
```

If you start the server with this option, it creates a key cache named `hot_cache` with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysqld --key_buffer_size=256K \  
            --extra_cache.key_buffer_size=128K \  
            --extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written `--default.key_buffer_size=256K`.) In addition, the server creates a second key cache named `extra_cache` that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysqld --key_buffer_size=6M \  
            --hot_cache.key_buffer_size=2M \  
            --cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named `hot_cache` to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;  
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a `LIKE` pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a

simple variable name may occur.

5.2.3.2. Dynamic System Variables

Many server system variables are dynamic and can be set at runtime using `SET GLOBAL` or `SET SESSION`. You can also obtain their values using `SELECT`. See [Section 5.2.3, “Using System Variables”](#).

The following table shows the full list of all dynamic system variables. The last column indicates for each variable whether `GLOBAL` or `SESSION` (or both) apply. The table also lists session options that can be set with the `SET` statement. [Section 13.5.3, “SET Syntax”](#), discusses these options.

Variables that have a type of “string” take a string value. Variables that have a type of “numeric” take a numeric value. Variables that have a type of “boolean” can be set to 0, 1, `ON` or `OFF`. (If you set them on the command line or in an option file, use the numeric values.) Variables that are marked as “enumeration” normally should be set to one of the available values for the variable, but can also be set to the number that corresponds to the desired enumeration value. For enumerated system variables, the first enumeration value corresponds to 0. This differs from `ENUM` columns, for which the first enumeration value corresponds to 1.

Variable Name	Value Type	Type
<code>autocommit</code>	boolean	SESSION
<code>big_tables</code>	boolean	SESSION
<code>binlog_cache_size</code>	numeric	GLOBAL
<code>bulk_insert_buffer_size</code>	numeric	GLOBAL SESSION
<code>character_set_client</code>	string	GLOBAL SESSION
<code>character_set_connection</code>	string	GLOBAL SESSION
<code>character_set_filesystem</code>	string	GLOBAL SESSION
<code>character_set_results</code>	string	GLOBAL SESSION
<code>character_set_server</code>	string	GLOBAL SESSION
<code>collation_connection</code>	string	GLOBAL SESSION
<code>collation_server</code>	string	GLOBAL SESSION

completion_type	numeric	GLOBAL SESSION
concurrent_insert	numeric	GLOBAL
connect_timeout	numeric	GLOBAL
default_week_format	numeric	GLOBAL SESSION
delay_key_write	OFF ON ALL	GLOBAL
delayed_insert_limit	numeric	GLOBAL
delayed_insert_timeout	numeric	GLOBAL
delayed_queue_size	numeric	GLOBAL
div_precision_increment	numeric	GLOBAL SESSION
engine_condition_pushdown	boolean	GLOBAL SESSION
error_count	numeric	SESSION
expire_logs_days	numeric	GLOBAL
flush	boolean	GLOBAL
flush_time	numeric	GLOBAL
foreign_key_checks	boolean	SESSION
ft_boolean_syntax	numeric	GLOBAL
group_concat_max_len	numeric	GLOBAL SESSION
identity	numeric	SESSION
innodb_autoextend_increment	numeric	GLOBAL
innodb_commit_concurrency	numeric	GLOBAL
innodb_concurrency_tickets	numeric	GLOBAL
innodb_max_dirty_pages_pct	numeric	GLOBAL
innodb_max_purge_lag	numeric	GLOBAL
innodb_support_xa	boolean	GLOBAL SESSION
innodb_sync_spin_loops	numeric	GLOBAL
innodb_table_locks	boolean	GLOBAL SESSION
innodb_thread_concurrency	numeric	GLOBAL
innodb_thread_sleep_delay	numeric	GLOBAL
insert_id	boolean	SESSION
interactive_timeout	numeric	GLOBAL SESSION
join_buffer_size	numeric	GLOBAL SESSION

key_buffer_size	numeric	GLOBAL
last_insert_id	numeric	SESSION
local_infile	boolean	GLOBAL
log_queries_not_using_indexes	boolean	GLOBAL
log_warnings	numeric	GLOBAL
long_query_time	numeric	GLOBAL SESSION
low_priority_updates	boolean	GLOBAL SESSION
max_allowed_packet	numeric	GLOBAL SESSION
max_binlog_cache_size	numeric	GLOBAL
max_binlog_size	numeric	GLOBAL
max_connect_errors	numeric	GLOBAL
max_connections	numeric	GLOBAL
max_delayed_threads	numeric	GLOBAL
max_error_count	numeric	GLOBAL SESSION
max_heap_table_size	numeric	GLOBAL SESSION
max_insert_delayed_threads	numeric	GLOBAL
max_join_size	numeric	GLOBAL SESSION
max_prepared_stmt_count	numeric	GLOBAL
max_relay_log_size	numeric	GLOBAL
max_seeks_for_key	numeric	GLOBAL SESSION
max_sort_length	numeric	GLOBAL SESSION
max_tmp_tables	numeric	GLOBAL SESSION
max_user_connections	numeric	GLOBAL
max_write_lock_count	numeric	GLOBAL
myisam_stats_method	enum	GLOBAL SESSION
multi_read_range	numeric	GLOBAL SESSION
myisam_data_pointer_size	numeric	GLOBAL
log_bin_trust_function_creators	boolean	GLOBAL
myisam_max_sort_file_size	numeric	GLOBAL SESSION
myisam_repair_threads	numeric	GLOBAL SESSION
myisam_sort_buffer_size	numeric	GLOBAL SESSION

net_buffer_length	numeric	GLOBAL SESSION
net_read_timeout	numeric	GLOBAL SESSION
net_retry_count	numeric	GLOBAL SESSION
net_write_timeout	numeric	GLOBAL SESSION
old_passwords	numeric	GLOBAL SESSION
optimizer_prune_level	numeric	GLOBAL SESSION
optimizer_search_depth	numeric	GLOBAL SESSION
preload_buffer_size	numeric	GLOBAL SESSION
query_alloc_block_size	numeric	GLOBAL SESSION
query_cache_limit	numeric	GLOBAL
query_cache_size	numeric	GLOBAL
query_cache_type	enumeration	GLOBAL SESSION
query_cache_wlock_invalidate	boolean	GLOBAL SESSION
query_prealloc_size	numeric	GLOBAL SESSION
range_alloc_block_size	numeric	GLOBAL SESSION
read_buffer_size	numeric	GLOBAL SESSION
read_only	numeric	GLOBAL
read_rnd_buffer_size	numeric	GLOBAL SESSION
rpl_recovery_rank	numeric	GLOBAL
safe_show_database	boolean	GLOBAL
secure_auth	boolean	GLOBAL
server_id	numeric	GLOBAL
slave_compressed_protocol	boolean	GLOBAL
slave_net_timeout	numeric	GLOBAL
slave_transaction_retries	numeric	GLOBAL
slow_launch_time	numeric	GLOBAL
sort_buffer_size	numeric	GLOBAL SESSION
sql_auto_is_null	boolean	SESSION
sql_big_selects	boolean	SESSION
sql_big_tables	boolean	SESSION
sql_buffer_result	boolean	SESSION

sql_log_bin	boolean	SESSION
sql_log_off	boolean	SESSION
sql_log_update	boolean	SESSION
sql_low_priority_updates	boolean	GLOBAL SESSION
sql_max_join_size	numeric	GLOBAL SESSION
sql_mode	enumeration	GLOBAL SESSION
sql_notes	boolean	SESSION
sql_quote_show_create	boolean	SESSION
sql_safe_updates	boolean	SESSION
sql_select_limit	numeric	SESSION
sql_slave_skip_counter	numeric	GLOBAL
updatable_views_with_limit	enumeration	GLOBAL SESSION
sql_warnings	boolean	SESSION
sync_binlog	numeric	GLOBAL
sync_frm	boolean	GLOBAL
storage_engine	enumeration	GLOBAL SESSION
table_cache	numeric	GLOBAL
table_type	enumeration	GLOBAL SESSION
thread_cache_size	numeric	GLOBAL
time_zone	string	GLOBAL SESSION
timestamp	boolean	SESSION
tmp_table_size	enumeration	GLOBAL SESSION
transaction_alloc_block_size	numeric	GLOBAL SESSION
transaction_prealloc_size	numeric	GLOBAL SESSION
tx_isolation	enumeration	GLOBAL SESSION
unique_checks	boolean	SESSION
wait_timeout	numeric	GLOBAL SESSION
warning_count	numeric	SESSION

5.2.4. Server Status Variables

The server maintains many status variables that provide information about its operation. You can view these variables and their values by using the SHOW STATUS statement:

```
mysql> SHOW STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Aborted_clients | 0 |
| Aborted_connects | 0 |
| Bytes_received | 155372598 |
| Bytes_sent | 1176560426 |
...
| Connections | 30023 |
| Created_tmp_disk_tables | 0 |
| Created_tmp_files | 3 |
| Created_tmp_tables | 2 |
...
| Threads_created | 217 |
| Threads_running | 88 |
| Uptime | 1389872 |
+-----+-----+
```

Many status variables are reset to 0 by the FLUSH STATUS statement.

The status variables have the following meanings. Variables with no version indicated were already present prior to MySQL 5.0. For information regarding their implementation history, see *MySQL 3.23, 4.0, 4.1 Reference Manual*.

- Aborted_clients

The number of connections that were aborted because the client died without closing the connection properly. See [Section A.2.10, “Communication Errors and Aborted Connections”](#).

- Aborted_connects

The number of failed attempts to connect to the MySQL server. See [Section A.2.10, “Communication Errors and Aborted Connections”](#).

- Binlog_cache_disk_use

The number of transactions that used the temporary binary log cache but

that exceeded the value of `binlog_cache_size` and used a temporary file to store statements from the transaction.

- `Binlog_cache_use`

The number of transactions that used the temporary binary log cache.

- `Bytes_received`

The number of bytes received from all clients.

- `Bytes_sent`

The number of bytes sent to all clients.

- `Com_xxx`

The `Com_xxx` statement counter variables indicate the number of times each `xxx` statement has been executed. There is one status variable for each type of statement. For example, `Com_delete` and `Com_insert` count `DELETE` and `INSERT` statements, respectively.

All of the `Com_stmt_xxx` variables are increased even if a prepared statement argument is unknown or an error occurred during execution. In other words, their values correspond to the number of requests issued, not to the number of requests successfully completed.

The `Com_stmt_xxx` status variables were added in 5.0.8:

- `Com_stmt_prepare`
- `Com_stmt_execute`
- `Com_stmt_fetch`
- `Com_stmt_send_long_data`
- `Com_stmt_reset`
- `Com_stmt_close`

Those variables stand for prepared statement commands. Their names refer to the COM_xxx command set used in the network layer. In other words, their values increase whenever prepared statement API calls such as **mysql_stmt_prepare()**, **mysql_stmt_execute()**, and so forth are executed. However, Com_stmt_prepare, Com_stmt_execute and Com_stmt_close also increase for PREPARE, EXECUTE, or DEALLOCATE PREPARE, respectively. Additionally, the values of the older (available since MySQL 4.1.3) statement counter variables Com_prepare_sql, Com_execute_sql, and Com_dealloc_sql increase for the PREPARE, EXECUTE, and DEALLOCATE PREPARE statements. Com_stmt_fetch stands for the total number of network round-trips issued when fetching from cursors.

- Compression

Whether the client connection uses compression in the client/server protocol. Added in MySQL 5.0.16.

- Connections

The number of connection attempts (successful or not) to the MySQL server.

- Created_tmp_disk_tables

The number of temporary tables on disk created automatically by the server while executing statements.

- Created_tmp_files

How many temporary files **mysqld** has created.

- Created_tmp_tables

The number of in-memory temporary tables created automatically by the server while executing statements. If Created_tmp_disk_tables is large, you may want to increase the tmp_table_size value to cause temporary tables to be memory-based instead of disk-based.

- Delayed_errors

The number of rows written with `INSERT DELAYED` for which some error occurred (probably duplicate key).

- `Delayed_insert_threads`

The number of `INSERT DELAYED` handler threads in use.

- `Delayed_writes`

The number of `INSERT DELAYED` rows written.

- `Flush_commands`

The number of executed `FLUSH` statements.

- `Handler_commit`

The number of internal `COMMIT` statements.

- `Handler_delete`

The number of times that rows have been deleted from tables.

- `Handler_discover`

The MySQL server can ask the `NDB Cluster` storage engine if it knows about a table with a given name. This is called discovery.

`Handler_discover` indicates the number of times that tables have been discovered via this mechanism.

- `Handler_prepare`

A counter for the prepare phase of two-phase commit operations. Added in MySQL 5.0.3.

- `Handler_read_first`

The number of times the first entry was read from an index. If this value is high, it suggests that the server is doing a lot of full index scans; for example, `SELECT col1 FROM foo`, assuming that `col1` is indexed.

- `Handler_read_key`

The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- `Handler_read_next`

The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- `Handler_read_prev`

The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`.

- `Handler_read_rnd`

The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that don't use keys properly.

- `Handler_read_rnd_next`

The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback`

The number of requests for a storage engine to perform a rollback operation.

- `Handler_savepoint`

The number of requests for a storage engine to place a savepoint. Added in MySQL 5.0.3.

- `Handler_savepoint_rollback`

The number of requests for a storage engine to roll back to a savepoint. Added in MySQL 5.0.3.

- `Handler_update`

The number of requests to update a row in a table.

- `Handler_write`

The number of requests to insert a row in a table.

- `Innodb_buffer_pool_pages_data`

The number of pages containing data (dirty or clean). Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_dirty`

The number of pages currently dirty. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_flushed`

The number of buffer pool page-flush requests. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_free`

The number of free pages. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_latched`

The number of latched pages in InnoDB buffer pool. These are pages currently being read or written or that cannot be flushed or removed for some other reason. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_misc`

The number of pages that are busy because they have been allocated for administrative overhead such as row locks or the adaptive hash index. This value can also be calculated as `Innodb_buffer_pool_pages_total` –

Innodb_buffer_pool_pages_free – Innodb_buffer_pool_pages_data.
Added in MySQL 5.0.2.

- Innodb_buffer_pool_pages_total

The total size of buffer pool, in pages. Added in MySQL 5.0.2.

- Innodb_buffer_pool_read_ahead_rnd

The number of “random” read-aheads initiated by InnoDB. This happens when a query scans a large portion of a table but in random order. Added in MySQL 5.0.2.

- Innodb_buffer_pool_read_ahead_seq

The number of sequential read-aheads initiated by InnoDB. This happens when InnoDB does a sequential full table scan. Added in MySQL 5.0.2.

- Innodb_buffer_pool_read_requests

The number of logical read requests InnoDB has done. Added in MySQL 5.0.2.

- Innodb_buffer_pool_reads

The number of logical reads that InnoDB could not satisfy from the buffer pool and had to do a single-page read. Added in MySQL 5.0.2.

- Innodb_buffer_pool_wait_free

Normally, writes to the InnoDB buffer pool happen in the background. However, if it is necessary to read or create a page and no clean pages are available, it is also necessary to wait for pages to be flushed first. This counter counts instances of these waits. If the buffer pool size has been set properly, this value should be small. Added in MySQL 5.0.2.

- Innodb_buffer_pool_write_requests

The number writes done to the InnoDB buffer pool. Added in MySQL 5.0.2.

- Innodb_data_fsyncs

The number of `fsync()` operations so far. Added in MySQL 5.0.2.

- `InnoDB_data_pending_fsyncs`

The current number of pending `fsync()` operations. Added in MySQL 5.0.2.

- `InnoDB_data_pending_reads`

The current number of pending reads. Added in MySQL 5.0.2.

- `InnoDB_data_pending_writes`

The current number of pending writes. Added in MySQL 5.0.2.

- `InnoDB_data_read`

The amount of data read so far, in bytes. Added in MySQL 5.0.2.

- `InnoDB_data_reads`

The total number of data reads. Added in MySQL 5.0.2.

- `InnoDB_data_writes`

The total number of data writes. Added in MySQL 5.0.2.

- `InnoDB_data_written`

The amount of data written so far, in bytes. Added in MySQL 5.0.2.

- `InnoDB_dblwr_writes`, `InnoDB_dblwr_pages_written`

The number of doublewrite operations that have been performed and the number of pages that have been written for this purpose. Added in MySQL 5.0.2. See [Section 14.2.14.1, “InnoDB Disk I/O”](#).

- `InnoDB_log_waits`

The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing. Added in MySQL 5.0.2.

- `Innodb_log_write_requests`

The number of log write requests. Added in MySQL 5.0.2.

- `Innodb_log_writes`

The number of physical writes to the log file. Added in MySQL 5.0.2.

- `Innodb_os_log_fsyncs`

The number of `fsync()` writes done to the log file. Added in MySQL 5.0.2.

- `Innodb_os_log_pending_fsyncs`

The number of pending log file `fsync()` operations. Added in MySQL 5.0.2.

- `Innodb_os_log_pending_writes`

The number of pending log file writes. Added in MySQL 5.0.2.

- `Innodb_os_log_written`

The number of bytes written to the log file. Added in MySQL 5.0.2.

- `Innodb_page_size`

The compiled-in InnoDB page size (default 16KB). Many values are counted in pages; the page size allows them to be easily converted to bytes. Added in MySQL 5.0.2.

- `Innodb_pages_created`

The number of pages created. Added in MySQL 5.0.2.

- `Innodb_pages_read`

The number of pages read. Added in MySQL 5.0.2.

- `Innodb_pages_written`

The number of pages written. Added in MySQL 5.0.2.

- `Innodb_row_lock_current_waits`

The number of row locks currently being waited for. Added in MySQL 5.0.3.

- `Innodb_row_lock_time`

The total time spent in acquiring row locks, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_time_avg`

The average time to acquire a row lock, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_time_max`

The maximum time to acquire a row lock, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_waits`

The number of times a row lock had to be waited for. Added in MySQL 5.0.3.

- `Innodb_rows_deleted`

The number of rows deleted from InnoDB tables. Added in MySQL 5.0.2.

- `Innodb_rows_inserted`

The number of rows inserted into InnoDB tables. Added in MySQL 5.0.2.

- `Innodb_rows_read`

The number of rows read from InnoDB tables. Added in MySQL 5.0.2.

- `Innodb_rows_updated`

The number of rows updated in InnoDB tables. Added in MySQL 5.0.2.

- `Key_blocks_not_flushed`

The number of key blocks in the key cache that have changed but have not yet been flushed to disk.

- `Key_blocks_unused`

The number of unused blocks in the key cache. You can use this value to determine how much of the key cache is in use; see the discussion of `key_buffer_size` in [Section 5.2.2, “Server System Variables”](#).

- `Key_blocks_used`

The number of used blocks in the key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- `Key_read_requests`

The number of requests to read a key block from the cache.

- `Key_reads`

The number of physical reads of a key block from disk. If `Key_reads` is large, then your `key_buffer_size` value is probably too small. The cache miss rate can be calculated as $\text{Key_reads}/\text{Key_read_requests}$.

- `Key_write_requests`

The number of requests to write a key block to the cache.

- `Key_writes`

The number of physical writes of a key block to disk.

- `Last_query_cost`

The total cost of the last compiled query as computed by the query optimizer. This is useful for comparing the cost of different query plans for

the same query. The default value of 0 means that no query has been compiled yet. This variable was added in MySQL 5.0.1, with a default value of -1. In MySQL 5.0.7, the default was changed to 0; also in version 5.0.7, the scope of `Last_query_cost` was changed to session rather than global.

Prior to MySQL 5.0.16, this variable was not updated for queries served from the query cache.

- `Max_used_connections`

The maximum number of connections that have been in use simultaneously since the server started.

- `Ndb_cluster_node_id`

If the server is acting as a MySQL Cluster node, then the value of this variable is its node ID in the cluster.

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

- `Ndb_config_from_host`

If the server is part of a MySQL Cluster, the value of this variable is the hostname or IP address of the Cluster management server from which it gets its configuration data.

If the server is not part of a MySQL Cluster, then the value of this variable is an empty string.

Prior to MySQL 5.0.23, this variable was named `Ndb_connected_host`.

- `Ndb_config_from_port`

If the server is part of a MySQL Cluster, the value of this variable is the number of the port through which it is connected to the Cluster management server from which it gets its configuration data.

If the server is not part of a MySQL Cluster, then the value of this

variable is 0.

Prior to MySQL 5.0.23, this variable was named `Ndb_connected_port`.

- `Ndb_number_of_storage_nodes`

If the server is part of a MySQL Cluster, the value of this variable is the number of data nodes in the cluster.

If the server is not part of a MySQL Cluster, then the value of this variable is 0.

- `Not_flushed_delayed_rows`

The number of rows waiting to be written in `INSERT DELAY` queues.

- `Open_files`

The number of files that are open.

- `Open_streams`

The number of streams that are open (used mainly for logging).

- `Open_tables`

The number of tables that are open.

- `Opened_tables`

The number of tables that have been opened. If `opened_tables` is big, your `table_cache` value is probably too small.

- `Qcache_free_blocks`

The number of free memory blocks in the query cache.

- `Qcache_free_memory`

The amount of free memory for the query cache.

- `Qcache_hits`

The number of query cache hits.

- `Qcache_inserts`

The number of queries added to the query cache.

- `Qcache_lowmem_prunes`

The number of queries that were deleted from the query cache because of low memory.

- `Qcache_not_cached`

The number of non-cached queries (not cacheable, or not cached due to the `query_cache_type` setting).

- `Qcache_queries_in_cache`

The number of queries registered in the query cache.

- `Qcache_total_blocks`

The total number of blocks in the query cache.

- `Questions`

The number of statements that clients have sent to the server.

- `Rpl_status`

The status of fail-safe replication (not yet implemented).

- `Select_full_join`

The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables.

- `Select_full_range_join`

The number of joins that used a range search on a reference table.

- `Select_range`

The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

- `Select_range_check`

The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

- `Select_scan`

The number of joins that did a full scan of the first table.

- `Slave_open_temp_tables`

The number of temporary tables that the slave SQL thread currently has open.

- `Slave_running`

This is ON if this server is a slave that is connected to a master.

- `Slave_retried_transactions`

The total number of times since startup that the replication slave SQL thread has retried transactions. This variable was added in version 5.0.4.

- `Slow_launch_threads`

The number of threads that have taken more than `slow_launch_time` seconds to create.

- `Slow_queries`

The number of queries that have taken more than `long_query_time` seconds. See [Section 5.12.4, “The Slow Query Log”](#).

- `Sort_merge_passes`

The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the `sort_buffer_size` system variable.

- `Sort_range`

The number of sorts that were done using ranges.

- `Sort_rows`

The number of sorted rows.

- `Sort_scan`

The number of sorts that were done by scanning the table.

- `Ssl_xxx`

Variables used for SSL connections.

- `Table_locks_immediate`

The number of times that a table lock was acquired immediately.

- `Table_locks_waited`

The number of times that a table lock could not be acquired immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

- `Threads_cached`

The number of threads in the thread cache.

- `Threads_connected`

The number of currently open connections.

- `Threads_created`

The number of threads created to handle connections. If `Threads_created` is big, you may want to increase the `thread_cache_size` value. The cache miss rate can be calculated as `Threads_created/Connections`.

- `Threads_running`

The number of threads that are not sleeping.

- `Uptime`

The number of seconds that the server has been up.

5.2.5. The Server SQL Mode

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients. This capability enables each application to tailor the server's operating mode to its own requirements.

Modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting **mysqld** with the `--sql-mode="modes"` option. *modes* is a list of different modes separated by comma (',') characters. The default value is empty (no modes set). The *modes* value also can be empty (`--sql-mode=""`) if you want to clear it explicitly.

You can change the SQL mode at runtime by using a `SET [GLOBAL|SESSION] sql_mode='modes'` statement to set the `sql_mode` system value. Setting the GLOBAL variable requires the SUPER privilege and affects the operation of all clients that connect from that time on. Setting the SESSION variable affects only the current client. Any client can change its own session `sql_mode` value at any time.

You can retrieve the current global or session `sql_mode` value with the following statements:

```
SELECT @@global.sql_mode;  
SELECT @@session.sql_mode;
```

The most important `sql_mode` values are probably these:

- `ANSI`

Change syntax and behavior to be more conformant to standard SQL.

- `STRICT_TRANS_TABLES`

If a value could not be inserted as given into a transactional table, abort the statement. For a non-transactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More detail is given later in this section. (Implemented in MySQL 5.0.2)

- `TRADITIONAL`

Make MySQL behave like a “traditional” SQL database system. A simple description of this mode is “give an error instead of a warning” when inserting an incorrect value into a column. **Note:** The `INSERT/UPDATE` aborts as soon as the error is noticed. This may not be what you want if you are using a non-transactional storage engine, because data changes made prior to the error are not be rolled back, resulting in a “partially done” update. (Added in MySQL 5.0.2)

When this manual refers to “strict mode,” it means a mode where at least one of `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` is enabled.

The following list describes all supported modes:

- `ALLOW_INVALID_DATES`

Don't do full checking of dates. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This is very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation). This mode applies to `DATE` and `DATETIME` columns. It does not apply `TIMESTAMP` columns, which always require a valid date.

This mode is implemented in MySQL 5.0.2. Before 5.0.2, this was the

default MySQL date-handling mode. As of 5.0.2, the server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as '2004-04-31' are converted to '0000-00-00' and a warning is generated. With strict mode enabled, invalid dates generate an error. To allow such dates, enable `ALLOW_INVALID_DATES`.

- `ANSI_QUOTES`

Treat `"` as an identifier quote character (like the ``` quote character) and not as a string quote character. You can still use ``` to quote identifiers with this mode enabled. With `ANSI_QUOTES` enabled, you cannot use double quotes to quote literal strings, because it is interpreted as an identifier.

- `ERROR_FOR_DIVISION_BY_ZERO`

Produce an error in strict mode (otherwise a warning) when a division by zero (or `MOD(x, 0)`) occurs during an `INSERT` or `UPDATE`. If this mode is not enabled, MySQL instead returns `NULL` for divisions by zero. For `INSERT IGNORE` or `UPDATE IGNORE`, MySQL generates a warning for divisions by zero, but the result of the operation is `NULL`. (Implemented in MySQL 5.0.2)

- `HIGH_NOT_PRECEDENCE`

From MySQL 5.0.2 on, the precedence of the `NOT` operator is such that expressions such as `NOT a BETWEEN b AND c` are parsed as `NOT (a BETWEEN b AND c)`. Before MySQL 5.0.2, the expression is parsed as `(NOT a) BETWEEN b AND c`. The old higher-precedence behavior can be obtained by enabling the `HIGH_NOT_PRECEDENCE` SQL mode. (Added in MySQL 5.0.2)

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 0
mysql> SET sql_mode = 'broken_not';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 1
```

- `IGNORE_SPACE`

Allow spaces between a function name and the `(` character. This forces all

function names to be treated as reserved words. As a result, if you want to access any database, table, or column name that is a reserved word, you must quote it. For example, because there is a `USER()` function, the name of the `user` table in the `mysql` database and the `User` column in that table become reserved, so you must quote them:

```
SELECT "User" FROM mysql."user";
```

The `IGNORE_SPACE` SQL mode applies to built-in functions, not to stored routines. It is always allowable to have spaces after a routine name, regardless of whether `IGNORE_SPACE` is enabled.

- `NO_AUTO_CREATE_USER`

Prevent `GRANT` from automatically creating new users if it would otherwise do so, unless a non-empty password also is specified. (Added in MySQL 5.0.2)

- `NO_AUTO_VALUE_ON_ZERO`

`NO_AUTO_VALUE_ON_ZERO` affects handling of `AUTO_INCREMENT` columns. Normally, you generate the next sequence number for the column by inserting either `NULL` or `0` into it. `NO_AUTO_VALUE_ON_ZERO` suppresses this behavior for `0` so that only `NULL` generates the next sequence number.

This mode can be useful if `0` has been stored in a table's `AUTO_INCREMENT` column. (Storing `0` is not a recommended practice, by the way.) For example, if you dump the table with **`mysqldump`** and then reload it, MySQL normally generates new sequence numbers when it encounters the `0` values, resulting in a table with contents different from the one that was dumped. Enabling `NO_AUTO_VALUE_ON_ZERO` before reloading the dump file solves this problem. **`mysqldump`** now automatically includes in its output a statement that enables `NO_AUTO_VALUE_ON_ZERO`, to avoid this problem.

- `NO_BACKSLASH_ESCAPES`

Disable the use of the backslash character (`\`) as an escape character within strings. With this mode enabled, backslash becomes an ordinary character like any other. (Implemented in MySQL 5.0.1)

- NO_DIR_IN_CREATE

When creating a table, ignore all INDEX DIRECTORY and DATA DIRECTORY directives. This option is useful on slave replication servers.

- NO_ENGINE_SUBSTITUTION

Prevents automatic substitution of the default storage engine when a statement such as CREATE TABLE specifies a storage engine that is disabled or not compiled in. (Implemented in MySQL 5.0.8)

- NO_FIELD_OPTIONS

Do not print MySQL-specific column options in the output of SHOW CREATE TABLE. This mode is used by **mysqldump** in portability mode.

- NO_KEY_OPTIONS

Do not print MySQL-specific index options in the output of SHOW CREATE TABLE. This mode is used by **mysqldump** in portability mode.

- NO_TABLE_OPTIONS

Do not print MySQL-specific table options (such as ENGINE) in the output of SHOW CREATE TABLE. This mode is used by **mysqldump** in portability mode.

- NO_UNSIGNED_SUBTRACTION

In integer subtraction operations, do not mark the result as UNSIGNED if one of the operands is unsigned. Note that this makes BIGINT UNSIGNED not 100% usable in all contexts. See [Section 12.8, “Cast Functions and Operators”](#).

```
mysql>t; SET sql_mode = '';
mysql>t; SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
|      18446744073709551615 |
+-----+
mysql>t; SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
```

```
mysql>t; SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
|                          -1 |
+-----+
```

- NO_ZERO_DATE

In strict mode, don't allow '0000-00-00' as a valid date. You can still insert zero dates with the IGNORE option. When not in strict mode, the date is accepted but a warning is generated. (Added in MySQL 5.0.2)

- NO_ZERO_IN_DATE

In strict mode, don't accept dates where the month or day part is 0. If used with the IGNORE option, MySQL inserts a '0000-00-00' date for any such date. When not in strict mode, the date is accepted but a warning is generated. (Added in MySQL 5.0.2)

- ONLY_FULL_GROUP_BY

Do not allow queries for which the SELECT list refers to non-aggregated columns that are not named in the GROUP BY clause. The following query is invalid with this mode enabled because address is not named in the GROUP BY clause:

```
SELECT name, address, MAX(age) FROM t GROUP BY name;
```

As of MySQL 5.0.23, this mode also restricts references to non-aggregated columns in the HAVING clause that are not named in the GROUP BY clause.

- PIPES_AS_CONCAT

Treat || as a string concatenation operator (same as CONCAT()) rather than as a synonym for OR.

- REAL_AS_FLOAT

Treat REAL as a synonym for FLOAT. By default, MySQL treats REAL as a synonym for DOUBLE.

- STRICT_ALL_TABLES

Enable strict mode for all storage engines. Invalid data values are rejected. Additional detail follows. (Added in MySQL 5.0.2)

- STRICT_TRANS_TABLES

Enable strict mode for transactional storage engines, and when possible for non-transactional storage engines. Additional details follow. (Implemented in MySQL 5.0.2)

Strict mode controls how MySQL handles input values that are invalid or missing. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a column that has no explicit DEFAULT clause in its definition.

For transactional tables, an error occurs for invalid or missing values in a statement when either of the STRICT_ALL_TABLES or STRICT_TRANS_TABLES modes are enabled. The statement is aborted and rolled back.

For non-transactional tables, the behavior is the same for either mode, if the bad value occurs in the first row to be inserted or updated. The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict option is enabled:

- For STRICT_ALL_TABLES, MySQL returns an error and ignores the rest of the rows. However, in this case, the earlier rows still have been inserted or updated. This means that you might get a partial update, which might not be what you want. To avoid this, it's best to use single-row statements because these can be aborted without changing the table.
- For STRICT_TRANS_TABLES, MySQL converts an invalid value to the closest valid value for the column and insert the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and continues processing the statement. Implicit defaults are described in [Section 11.1.4, “Data Type Default Values”](#).

Strict mode disallows invalid date values such as '2004-04-31'. It does not disallow dates with zero parts such as '2004-04-00' or “zero” dates. To disallow these as well, enable the NO_ZERO_IN_DATE and NO_ZERO_DATE SQL modes in addition to strict mode.

If you are not using strict mode (that is, neither STRICT_TRANS_TABLES nor STRICT_ALL_TABLES is enabled), MySQL inserts adjusted values for invalid or missing values and produces warnings. In strict mode, you can produce this behavior by using INSERT IGNORE or UPDATE IGNORE. See [Section 13.5.4.25, “SHOW WARNINGS Syntax”](#).

The following special modes are provided as shorthand for combinations of mode values from the preceding list. All are available in MySQL 5.0 beginning with version 5.0.0, except for TRADITIONAL, which was implemented in MySQL 5.0.2.

The descriptions include all mode values that are available in the most recent version of MySQL. For older versions, a combination mode does not include individual mode values that are not available except in newer versions.

- ANSI

Equivalent to REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE. Before MySQL 5.0.3, ANSI also includes ONLY_FULL_GROUP_BY. See [Section 1.9.3, “Running MySQL in ANSI Mode”](#).

- DB2

Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MAXDB

Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- MSSQL

Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MYSQL323

Equivalent to NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

- MYSQL40

Equivalent to NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

- ORACLE

Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- POSTGRESQL

Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- TRADITIONAL

Equivalent to STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER.

5.2.6. The MySQL Server Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

Server shutdown can be initiated several ways. For example, a user with the SHUTDOWN privilege can execute a **mysqladmin shutdown** command. **mysqladmin** can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a SIGTERM signal. A server running as a service on Windows shuts down when the services manager

tells it to.

2. The server creates a shutdown thread if necessary.

Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

```
Error: Can't create thread to kill server
```

3. The server stops accepting new connections.

To prevent new activity from being initiated during shutdown, the server stops accepting new client connections. It does this by closing the network connections to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

For each thread that is associated with a client connection, the connection to the client is broken and the thread is marked as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see [Section 13.5.5.3, “KILL Syntax”](#), in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on MyISAM tables.

For threads that have an open transaction, the transaction is rolled back. Note that if a thread is updating a non-transactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated, because the operation can terminate before completion.

If the server is a master replication server, threads associated with currently connected slaves are treated like other client threads. That is, each one is

marked as killed and exits when it next checks its state.

If the server is a slave replication server, the I/O and SQL threads, if active, are stopped before client threads are marked as killed. The SQL thread is allowed to finish its current statement (to avoid causing replication problems), and then stops. If the SQL thread was in the middle of a transaction at this point, the transaction is rolled back.

5. Storage engines are shut down or closed.

At this stage, the table cache is flushed and all open tables are closed.

Each storage engine performs any actions necessary for tables that it manages. For example, MyISAM flushes any pending index writes for a table. InnoDB flushes its buffer pool to disk (starting from 5.0.5: unless `innodb_fast_shutdown` is 2), writes the current LSN to the tablespace, and terminates its own internal threads.

6. The server exits.

5.2.7. MySQL Server-Side Help Support

MySQL Server supports a `HELP` statement that returns online information from the MySQL Reference manual (see [Section 13.3.2, “HELP Syntax”](#)). The proper operation of this statement requires that the help tables in the `mysql` database be initialized with help topic information, which is done by processing the contents of the `fill_help_tables.sql` script.

For a MySQL binary distribution on Unix, help table setup occurs when you run **`mysql_install_db`**. For an RPM distribution on Linux or binary distribution on Windows, help table setup occurs as part of the MySQL installation process.

For a MySQL source distribution, you can find the `fill_help_tables.sql` file in the `scripts` directory. To load the file manually, make sure that you have initialized the `mysql` database by running **`mysql_install_db`**, and then process the file with the **`mysql`** client as follows:

```
shell> mysql -u root mysql < fill_help_tables.sql
```

If you are working with BitKeeper and a MySQL development source tree, the

tree doesn't contain `fill_help_tables.sql`. You can download the proper file for your version of MySQL from <http://dev.mysql.com/doc/>. After downloading and uncompressing the file, process it with **mysql** as just described.

5.3. The `mysqld-max` Extended MySQL Server

A MySQL-Max server is a version of the **mysqld** MySQL server that has been built to include additional features. The MySQL-Max distribution to use depends on your platform:

- For Windows, MySQL binary distributions include both the standard server (`mysqld.exe`) and the MySQL-Max server (**`mysqld-max.exe`**), so no special distribution is needed. Just use a regular Windows distribution. See [Section 2.3, “Installing MySQL on Windows”](#).
- For Linux, if you install MySQL using RPM distributions, the MySQL-Max RPM presupposes that you have already installed the regular server RPM. Use the regular MySQL-server RPM first to install a standard server named **mysqld**, and then use the MySQL-Max RPM to install a server named **mysqld-max**. See [Section 2.4, “Installing MySQL on Linux”](#), for more information on the Linux RPM packages.
- All other MySQL-Max distributions contain a single server that is named **mysqld** but that has the additional features included.

You can find the MySQL-Max binaries on the MySQL AB Web site at <http://dev.mysql.com/downloads/>.

MySQL AB builds the MySQL-Max servers by using the following **configure** options:

- `--with-server-suffix=-max`

This option adds a `-max` suffix to the **mysqld** version string.

- `--with-innodb`

This option enables support for the InnoDB storage engine. MySQL-Max servers always include InnoDB support. From MySQL 4.0 onward, InnoDB is included by default in all binary distributions, so a MySQL-Max server is not needed to obtain InnoDB support.

- `--with-bdb`

This option enables support for the Berkeley DB (BDB) storage engine on those platforms for which BDB is available. (See notes in the following discussion.)

- `--with-blackhole-storage-engine`

This option enables support for the BLACKHOLE storage engine.

- `--with-csv-storage-engine`

This option enables support for the CSV storage engine.

- `--with-example-storage-engine`

This option enables support for the EXAMPLE storage engine.

- `--with-federated-storage-engine`

This option enables support for the FEDERATED storage engine.

- `--with-ndbcluster`

This option enables support for the NDB Cluster storage engine on those platforms for which Cluster is available. (See notes in the following discussion.)

- `USE_SYMDIR`

This define is enabled to turn on database symbolic link support for Windows. From MySQL 4.0 onward, symbolic link support is enabled for all Windows servers, so a MySQL-Max server is not needed to take advantage of this feature.

MySQL-Max binary distributions are a convenience for those who wish to install precompiled programs. If you build MySQL using a source distribution, you can build your own Max-like server by enabling the same features at configuration time that the MySQL-Max binary distributions are built with.

MySQL-Max servers include the BerkeleyDB (BDB) storage engine whenever

possible, but not all platforms support BDB.

Currently, MySQL Cluster is supported on Linux (on most platforms), Solaris, Mac OS X, and HP-UX only. Some users have reported success in using MySQL Cluster built from source on BSD operating systems, but these are not officially supported at this time. Note that, even for servers compiled with Cluster support, the NDB Cluster storage engine is not enabled by default. You must start the server with the `--ndbcluster` option to use it as part of a MySQL Cluster. (For details, see [Section 15.4, “MySQL Cluster Configuration”](#).)

The following table shows the platforms for which MySQL-Max binaries include support for BDB and NDB Cluster.

System	BDB Support	NDB Support
AIX 5.2	N	N
HP-UX	Y	Y
Linux-IA-64	N	Y
Linux-Intel	Y	Y
Mac OS X	N	Y
NetWare	N	N
SCO 6	N	N
Solaris-SPARC	Y	Y
Solaris-Intel	N	Y
Solaris-AMD 64	Y	Y
Windows NT/2000/XP	Y	N

To find out which storage engines your server supports, use the `SHOW ENGINES` statement. (See [Section 13.5.4.10, “SHOW ENGINES Syntax”](#).) For example:

```
mysql> SHOW ENGINES\G
***** 1. row *****
Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
***** 2. row *****
Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
```

```

***** 3. row *****
Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
***** 4. row *****
Engine: BerkeleyDB
Support: NO
Comment: Supports transactions and page-level locking
***** 5. row *****
Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write to it disappea
...

```

The precise output from `SHOW ENGINES` may vary according to the MySQL version used (and the features that are enabled). The Support values in the output indicate the server's level of support for each feature, as shown here:

Value	Meaning
YES	The feature is supported and is active.
NO	The feature is not supported.
DISABLED	The feature is supported but has been disabled.

A value of `NO` means that the server was compiled without support for the feature, so it cannot be activated at runtime.

A value of `DISABLED` occurs either because the server was started with an option that disables the feature, or because not all options required to enable it were given. In the latter case, the error log file should contain a reason indicating why the option is disabled. See [Section 5.12.1, “The Error Log”](#).

You might also see `DISABLED` for a storage engine if the server was compiled to support it, but was started with a `--skip-engine` option. For example, `--skip-innodb` disables the InnoDB engine. For the NDB Cluster storage engine, `DISABLED` means the server was compiled with support for MySQL Cluster, but was not started with the `--ndb-cluster` option.

All MySQL servers support `myISAM` tables, because `myISAM` is the default storage engine.

5.4. MySQL Server Startup Programs

This section describes several programs that are used to start **mysqld**, the MySQL server.

5.4.1. **mysqld_safe** — MySQL Server Startup Script

mysqld_safe is the recommended way to start a **mysqld** server on Unix and NetWare. **mysqld_safe** adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log file. NetWare-specific behaviors are listed later in this section.

Note: To preserve backward compatibility with older versions of MySQL, MySQL binary distributions still include **safe_mysqld** as a symbolic link to **mysqld_safe**. However, you should not rely on this because it is removed as of MySQL 5.1.

By default, **mysqld_safe** tries to start an executable named **mysqld-max** if it exists, and **mysqld** otherwise. Be aware of the implications of this behavior:

- On Linux, the MySQL-Max RPM relies on this **mysqld_safe** behavior. The RPM installs an executable named **mysqld-max**, which causes **mysqld_safe** to automatically use that executable rather than **mysqld** from that point on.
- If you install a MySQL-Max distribution that includes a server named **mysqld-max**, and then upgrade later to a non-Max version of MySQL, **mysqld_safe** will still attempt to run the old **mysqld-max** server. If you perform such an upgrade, you should manually remove the old **mysqld-max** server to ensure that **mysqld_safe** runs the new **mysqld** server.

To override the default behavior and specify explicitly the name of the server you want to run, specify a `--mysqld` or `--mysqld-version` option to **mysqld_safe**. You can also use `--ledir` to indicate the directory where **mysqld_safe** should look for the server.

Many of the options to **mysqld_safe** are the same as the options to **mysqld**. See [Section 5.2.1, “mysqld Command Options”](#).

All options specified to **mysqld_safe** on the command line are passed to **mysqld**. If you want to use any options that are specific to **mysqld_safe** and that **mysqld** doesn't support, do not specify them on the command line. Instead, list them in the [mysqld_safe] group of an option file. See [Section 4.3.2, “Using Option Files”](#).

mysqld_safe reads all options from the [mysqld], [server], and [mysqld_safe] sections in option files. For backward compatibility, it also reads [safe_mysqld] sections, although you should rename such sections to [mysqld_safe] in MySQL 5.0 installations.

mysqld_safe supports the following options:

- --help

Display a help message and exit. (Added in MySQL 5.0.3)

- --autoclose

(NetWare only) On NetWare, **mysqld_safe** provides a screen presence. When you unload (shut down) the **mysqld_safe** NLM, the screen does not by default go away. Instead, it prompts for user input:

```
*<NLM has terminated; Press any key to close the screen>*
```

If you want NetWare to close the screen automatically instead, use the --autoclose option to **mysqld_safe**.

- --basedir=path

The path to the MySQL installation directory.

- --core-file-size=size

The size of the core file that **mysqld** should be able to create. The option value is passed to **ulimit -c**.

- --datadir=path

The path to the data directory.

- `--defaults-extra-file=path`

The name of an option file to be read in addition to the usual option files. This must be the first option on the command line if it is used. As of MySQL 5.0.6, if the file does not exist or is otherwise inaccessible, the server will exit with an error.

- `--defaults-file=file_name`

The name of an option file to be read instead of the usual option files. This must be the first option on the command line if it is used.

- `--ledir=path`

If **mysqld_safe** cannot find the server, use this option to indicate the pathname to the directory where the server is located.

- `--log-error=file_name`

Write the error log to the given file. See [Section 5.12.1, “The Error Log”](#).

- `--mysqld=prog_name`

The name of the server program (in the `ledir` directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution. If **mysqld_safe** cannot find the server, use the `--ledir` option to indicate the pathname to the directory where the server is located.

- `--mysqld-version=suffix`

This option is similar to the `--mysqld` option, but you specify only the suffix for the server program name. The basename is assumed to be **mysqld**. For example, if you use `--mysqld-version=max`, **mysqld_safe** starts the **mysqld-max** program in the `ledir` directory. If the argument to `--mysqld-version` is empty, **mysqld_safe** uses **mysqld** in the `ledir` directory.

- `--nice=priority`

Use the `nice` program to set the server's scheduling priority to the given value.

- `--no-defaults`

Do not read any option files. This must be the first option on the command line if it is used.

- `--open-files-limit=count`

The number of files that **mysqld** should be able to open. The option value is passed to **ulimit -n**. Note that you need to start **mysqld_safe** as root for this to work properly!

- `--pid-file=file_name`

The pathname of the process ID file.

- `--port=port_num`

The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the root system user.

- `--socket=path`

The Unix socket file that the server should use when listening for local connections.

- `--timezone=timezone`

Set the `TZ` time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- `--user={user_name|user_id}`

Run the **mysqld** server as the user having the name *user_name* or the numeric user ID *user_id*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

If you execute **mysqld_safe** with the `--defaults-file` or `--defaults-extra-` option option to name an option file, the option must be the first one given on the command line or the option file will not be used. For example, this command will not use the named option file:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

Instead, use the following command:

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

The **mysqld_safe** script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See [Section 2.1.5, “Installation Layouts”](#).) **mysqld_safe** expects one of the following conditions to be true:

- The server and databases can be found relative to the working directory (the directory from which **mysqld_safe** is invoked). For binary distributions, **mysqld_safe** looks under its working directory for `bin` and `data` directories. For source distributions, it looks for `libexec` and `var` directories. This condition should be met if you execute **mysqld_safe** from your MySQL installation directory (for example, `/usr/local/mysql` for a binary distribution).
- If the server and databases cannot be found relative to the working directory, **mysqld_safe** attempts to locate them by absolute pathnames. Typical locations are `/usr/local/libexec` and `/usr/local/var`. The actual locations are determined from the values configured into the distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because **mysqld_safe** tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run **mysqld_safe** from the MySQL installation directory:

```
shell> cd mysql_installation_directory  
shell> bin/mysqld_safe &
```

If **mysqld_safe** fails, even when invoked from the MySQL installation directory, you can specify the `--ledir` and `--datadir` options to indicate the directories in

which the server and databases are located on your system.

Normally, you should not edit the **mysqld_safe** script. Instead, configure **mysqld_safe** by using command-line options or options in the `[mysqld_safe]` section of a `my.cnf` option file. In rare cases, it might be necessary to edit **mysqld_safe** to get it to start the server properly. However, if you do this, your modified version of **mysqld_safe** might be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.

On NetWare, **mysqld_safe** is a NetWare Loadable Module (NLM) that is ported from the original Unix shell script. It starts the server as follows:

1. Runs a number of system and option checks.
2. Runs a check on MyISAM tables.
3. Provides a screen presence for the MySQL server.
4. Starts **mysqld**, monitors it, and restarts it if it terminates in error.
5. Sends error messages from **mysqld** to the `host_name.err` file in the data directory.
6. Sends **mysqld_safe** screen output to the `host_name.safe` file in the data directory.

5.4.2. **mysql.server** — MySQL Server Startup Script

MySQL distributions on Unix include a script named **mysql.server**. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the Mac OS X Startup Item for MySQL.

mysql.server can be found in the `support-files` directory under your MySQL installation directory or in a MySQL source distribution.

If you use the Linux server RPM package (`MySQL-server-VERSION.rpm`), the **mysql.server** script will be installed in the `/etc/init.d` directory with the name `mysql1`. You need not install it manually. See [Section 2.4, “Installing MySQL on](#)

[Linux](#)”, for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as **mysqld**.

If you install MySQL from a source distribution or using a binary distribution format that does not install **mysql.server** automatically, you can install it manually. Instructions are provided in [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).

mysql.server reads options from the [mysql.server] and [mysqld] sections of option files. For backward compatibility, it also reads [mysql_server] sections, although you should rename such sections to [mysql.server] when using MySQL 5.0.

5.4.3. **mysqld_multi** — Manage Multiple MySQL Servers

mysqld_multi is designed to manage several **mysqld** processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status. The MySQL Instance Manager is an alternative means of managing multiple servers (see [Section 5.5, “mysqlmanager — The MySQL Instance Manager”](#)).

mysqld_multi searches for groups named [mysqldN] in my.cnf (or in the file named by the --config-file option). *N* can be any positive integer. This number is referred to in the following discussion as the option group number, or *GNR*. Group numbers distinguish option groups from one another and are used as arguments to **mysqld_multi** to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the [mysqld] group used for starting **mysqld**. (See, for example, [Section 2.10.2.2, “Starting and Stopping MySQL Automatically”](#).) However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see [Section 5.13, “Running Multiple MySQL Servers on the Same Machine”](#).

To invoke **mysqld_multi**, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]
```

start, stop, and report indicate which operation to perform. You can perform the designated operation for a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, **mysqld_multi** performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named [mysqld17] is 17. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value 10-13 represents groups [mysqld10] through [mysqld13]. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group [mysqld17]:

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups [mysqld8] and [mysqld10] through [mysqld13]:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

mysqld_multi supports the following options:

- --help

Display a help message and exit.

- --config-file=file_name

Specify the name of an alternative option file. This affects where **mysqld_multi** looks for [mysqldN] option groups. Without this option, all options are read from the usual my.cnf file. The option does not affect where **mysqld_multi** reads its own options, which are always taken from the [mysqld_multi] group in the usual my.cnf file.

- `--example`

Display a sample option file.

- `--log=file_name`

Specify the name of the log file. If the file exists, log output is appended to it.

- `--mysqladmin=prog_name`

The **mysqladmin** binary to be used to stop servers.

- `--mysqld=prog_name`

The **mysqld** binary to be used. Note that you can specify **mysqld_safe** as the value for this option also. If you use **mysqld_safe** to start the server, you can include the `mysqld` or `ledir` options in the corresponding `[mysqldN]` option group. These options indicate the name of the server that **mysqld_safe** should start and the pathname of the directory where the server is located. (See the descriptions for these options in [Section 5.4.1, “mysqld_safe — MySQL Server Startup Script”](#).) Example:

```
[mysqld38]
mysqld = mysqld-max
ledir  = /opt/local/mysql/libexec
```

- `--no-log`

Print log information to `stdout` rather than to the log file. By default, output goes to the log file.

- `--password=password`

The password of the MySQL account to use when invoking **mysqladmin**. Note that the password value is not optional for this option, unlike for other MySQL programs.

- `--silent`

Silent mode; disable warnings.

- `--tcp-ip`

Connect to each MySQL server via the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only via the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects stop and report operations.

- `--user=user_name`

The username of the MySQL account to use when invoking **mysqladmin**.

- `--verbose`

Be more verbose.

- `--version`

Display version information and exit.

Some notes about **mysqld_multi**:

- **Most important:** Before using **mysqld_multi** be sure that you understand the meanings of the options that are passed to the **mysqld** servers and *why* you would want to have separate **mysqld** processes. Beware of the dangers of using multiple **mysqld** servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory does *not* give you extra performance in a threaded system. See [Section 5.13, “Running Multiple MySQL Servers on the Same Machine”](#).
- **Important:** Make sure that the data directory for each server is fully accessible to the Unix account that the specific **mysqld** process is started as. *Do not* use the Unix *root* account for this, unless you *know* what you are doing. See [Section 5.7.5, “How to Run MySQL as a Normal User”](#).
- Make sure that the MySQL account used for stopping the **mysqld** servers (with the **mysqladmin** program) has the same username and password for each server. Also, make sure that the account has the SHUTDOWN privilege. If the servers that you want to manage have different usernames or passwords for the administrative accounts, you might want to create an account on

each server that has the same username and password. For example, you might set up a common `multi_admin` account by executing the following commands for each server:

```
shell> mysql -u root -S /tmp/mysql.sock -p
Enter password:
mysql> GRANT SHUTDOWN ON *.*
      -> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
```

See [Section 5.8.2, “How the Privilege System Works”](#). You have to do this for each `mysqld` server. Change the connection parameters appropriately when connecting to each one. Note that the hostname part of the account name must allow you to connect as `multi_admin` from the host where you want to run `mysqld_multi`.

- The Unix socket file and the TCP/IP port number must be different for every `mysqld`.
- The `--pid-file` option is very important if you are using `mysqld_safe` to start `mysqld` (for example, `--mysqld=mysqld_safe`) Every `mysqld` should have its own process ID file. The advantage of using `mysqld_safe` instead of `mysqld` is that `mysqld_safe` monitors its `mysqld` process and restarts it if the process terminates due to a signal sent using `kill -9` or for other reasons, such as a segmentation fault. Please note that the `mysqld_safe` script might require that you start it from a certain place. This means that you might have to change location to a certain directory before running `mysqld_multi`. If you have problems starting, please see the `mysqld_safe` script. Check especially the lines:

```
-----
MY_PWD=`pwd`
# Check if we are starting this relative (for the binary release
if test -d $MY_PWD/data/mysql -a -f ./share/mysql/english/errmsg
-x ./bin/mysqld
-----
```

The test performed by these lines should be successful, or you might encounter problems. See [Section 5.4.1, “`mysqld_safe` — MySQL Server Startup Script”](#).

- You might want to use the `--user` option for `mysqld`, but to do this you need to run the `mysqld_multi` script as the Unix root user. Having the

option in the option file doesn't matter; you just get a warning if you are not the superuser and the **mysqld** processes are started under your own Unix account.

The following example shows how you might set up an option file for use with **mysqld_multi**. The order in which the **mysqld** programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth [mysqldN] groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen
```

```
[mysqld_multi]
mysqld      = /usr/local/bin/mysqld_safe
mysqladmin  = /usr/local/bin/mysqladmin
user        = multi_admin
password    = multipass

[mysqld2]
socket      = /tmp/mysql.sock2
port        = 3307
pid-file    = /usr/local/mysql/var2/hostname.pid2
datadir     = /usr/local/mysql/var2
language    = /usr/local/share/mysql/english
user        = john

[mysqld3]
socket      = /tmp/mysql.sock3
port        = 3308
pid-file    = /usr/local/mysql/var3/hostname.pid3
datadir     = /usr/local/mysql/var3
language    = /usr/local/share/mysql/swedish
user        = monty

[mysqld4]
socket      = /tmp/mysql.sock4
port        = 3309
pid-file    = /usr/local/mysql/var4/hostname.pid4
datadir     = /usr/local/mysql/var4
language    = /usr/local/share/mysql/estonia
user        = tonu

[mysqld6]
socket      = /tmp/mysql.sock6
```

```
port          = 3311
pid-file      = /usr/local/mysql/var6/hostname.pid6
datadir       = /usr/local/mysql/var6
language      = /usr/local/share/mysql/japanese
user          = jani
```

See [Section 4.3.2, “Using Option Files”](#).

5.5. **mysqlmanager** — The MySQL Instance Manager

mysqlmanager is the MySQL Instance Manager (IM). This program is a daemon running on a TCP/IP port that serves to monitor and manage MySQL Database Server instances. MySQL Instance Manager is available for Unix-like operating systems, and also on Windows as of MySQL 5.0.13.

MySQL Instance Manager is included in MySQL distributions from version 5.0.3, and can be used in place of the `mysqld_safe` script to start and stop the MySQL Server, *even from a remote host*. MySQL Instance Manager also implements the functionality (and most of the syntax) of the **mysqld_multi** script. A more detailed description of MySQL Instance Manager follows.

5.5.1. Starting the MySQL Server with MySQL Instance Manager

Normally, the **mysqld** MySQL Database Server is started with the **mysql.server** script, which usually resides in the `/etc/init.d/` folder. In MySQL 5.0.3, this script invokes **mysqlmanager** (the MySQL Instance Manager binary) to start MySQL. (In prior versions of MySQL the **mysqld_safe** script is used for this purpose.) Starting from MySQL 5.0.4, the behavior of the startup script was changed again to incorporate both setup schemes. In version 5.0.4, the startup script uses the old scheme (invoking **mysqld_safe**) by default, but one can set the `use_mysqld_safe` variable in the script to 0 (zero) to use the MySQL Instance Manager to start a server.

Starting with MySQL 5.0.19, you can instead modify the `my.cnf` file by adding `use-manager` to the `[mysql.server]` section:

```
[mysql.server]
use-manager
```

The Instance Manager's behavior in this case depends on the options given in the MySQL configuration file. If there is no configuration file, the MySQL Instance Manager creates a server instance named `mysqld` and attempts to start it with default (compiled-in) configuration values. This means that the IM cannot guess the placement of **mysqld** if it is not installed in the default location. If you have installed the MySQL server in a non-standard location, you should use a configuration file. See [Section 2.1.5, “Installation Layouts”](#).

If there is a configuration file, the IM reads it to find `[mysqldN]` sections (for example, `[mysqld1]`, `[mysqld2]`, and so forth). Each such section specifies an instance. When it starts, the Instance Manager attempts to start all server instances that it finds. By default, the Instance Manager stops all server instances when it shuts down.

Warning

The `[mysqld]` section name causes unpredictable results when used in conjunction with the Instance Manager. When using the Instance Manager, check that no section is named `[mysqld]`.

Note that there is a special `--mysqld-path=path-to-mysqld-binary` option that is recognized only by the IM. Use this variable to let the IM know where the **mysqld** binary resides. You should also set `basedir` and `datadir` options for the server.

The typical startup/shutdown cycle for a MySQL server with the MySQL Instance Manager enabled is as follows:

1. The MySQL Instance Manager is started with `/etc/init.d/mysql` script.
2. The MySQL Instance Manager starts all instances and monitors them.
3. If a server instance fails the MySQL Instance Manager restarts it.
4. If the MySQL Instance Manager is shut down (for instance with the `/etc/init.d/mysql stop` command), all instances are shut down by the MySQL Instance Manager.

5.5.2. Connecting to the MySQL Instance Manager and Creating User Accounts

Communication with the MySQL Instance Manager is handled using the MySQL client-server protocol. As such, you can connect to the IM using the standard **mysql** client program, as well as the MySQL C API. The IM supports the version of the MySQL client-server protocol used by the client tools and libraries distributed along with MySQL 4.1 or later.

5.5.2.1. Instance Manager Users and Passwords

The Instance Manager stores its user information in a password file. The default name of the password file is `/etc/mysqlmanager.password`.

Password entries have the following format:

```
petr:*35110DC9B4D8140F5DE667E28C72DD2597B5C848
```

If there are no entries in the `/etc/mysqlmanager.password` file, you cannot connect to the Instance Manager.

To generate a new entry, invoke Instance Manager with the `--passwd` option. Then the output can be appended to the `/etc/mysqlmanager.password` file to add a new user. Here is an example:

```
shell> mysqlmanager --passwd >> /etc/mysqlmanager.password
Creating record for new user.
Enter user name: mike
Enter password: password
Re-type password: password
```

The preceding command causes the following line to be added to `/etc/mysqlmanager.password`:

```
mike:*00A51F3F48415C7D4E8908980D443C29C69B60C9
```

Note

The Instance Manager must be restarted after adding/changing passwords.

5.5.2.2. MySQL Server Accounts for Status Monitoring

To monitor server status, the MySQL Instance Manager will attempt to connect to the MySQL server instance at regular intervals using the `MySQL_Instance_Manager@localhost` user account with a password of `check_connection`.

You are *not* required to create a `MySQL_Instance_M@localhost` user account in order for the MySQL Instance Manager to monitor server status, as a login

failure is sufficient to identify that the server is operational. However, if the account does not exist, failed connection attempts are logged by the server to its general query log (see [Section 5.12.2, “The General Query Log”](#)).

5.5.3. MySQL Instance Manager Command Options

The MySQL Instance Manager supports a number of command line options. For a brief listing, invoke **mysqlmanager** with the `--help` option.

mysqlmanager supports the following options:

- `--help, -?`

Display a help message and exit.

- `--angel-pid-file=file_name`

The file in which the angel process records its process ID when **mysqlmanager** runs in daemon mode. By default, this file is named `mysqlmanager.angel.pid`. If the `--pid-file` option is given, the default angel PID file becomes the same except that any extension is replaced with an extension of `.angel.pid`. This option was added in MySQL 5.0.23.

- `--bind-address=IP`

The IP address to bind to.

- `--default-mysqld-path=path`

On Unix, the pathname of the MySQL Server binary, if no path was provided in the instance section. Example: `--default-mysqld-path=/usr/sbin/mysqld`

- `--defaults-file=file_name`

Read Instance Manager and MySQL Server settings from the given file. All configuration changes by the Instance Manager will be made to this file. This must be the first option on the command line if it is used.

- `--install`

On Windows, install Instance Manager as a Windows service. This option was added in MySQL 5.0.11.

- `--log=file_name`

The path to the IM log file. This is used with the **--run-as-service** option.

- `--monitoring-interval=seconds`

The interval in seconds for monitoring instances. The default value is 20 seconds. Instance Manager tries to connect to each monitored instance using the non-existing `MySQL_Instance_Manager` user account to check whether it is alive/not hanging. In the case of a failure to connect, IM performs several attempts to restart the instance. The nonguarded option in the appropriate instance section disables this behavior for a particular instance. The monitoring process will produce messages in the general query log similar to the following:

```
Access denied for user 'MySQL_Instance_M'@'localhost' (using pas
```

- `--passwd, -P`

Prepare an entry for the password file and exit.

- `--password-file=file_name`

Look for the Instance Manager users and passwords in this file. The default file is `/etc/mysqlmanager.passwd`.

- `--pid-file=file_name`

The process ID file to use. By default, this file is named `mysqlmanager.pid`.

- `--port=port_num`

The TCP/IP port number to use for incoming connections. (The default port number assigned by IANA is 2273).

- `--print-defaults`

Print the current defaults and exit. This must be the first option on the command line if it is used.

- `--remove`

On Windows, removes Instance Manager as a Windows service. This assumes that Instance Manager has been run with `--install` previously. This option was added in MySQL 5.0.11.

- `--run-as-service`

On Unix, daemonize and start the angel process. The angel process is simple and unlikely to crash. It will restart the Instance Manager itself in case of a failure.

- `--socket=path`

On Unix, the socket file to use for incoming connections. By default, the file is named `/tmp/mysqlmanager.sock`.

- `--standalone`

On Windows, run Instance Manager in standalone mode. This option was added in MySQL 5.0.13.

- `--user=user_name`

On Unix, the username to start and run the **mysqlmanager** under. It is recommended to run **mysqlmanager** under the same user account used to run the **mysqld** server. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

- `--version, -V`

Output version information and exit.

- `--wait-timeout=N`

The number of seconds to wait for activity on a connection before closing it. The default is 28800 seconds (8 hours).

This option was added in MySQL 5.0.19. Before that, the timeout is 30 seconds and cannot be changed.

5.5.4. MySQL Instance Manager Configuration Files

Instance Manager uses the standard `my.cnf` file. It uses the `[manager]` section to read options for itself and the `[mysqldN]` sections to create instances. The `[manager]` section contains any of the options listed in [Section 5.5.3, “MySQL Instance Manager Command Options”](#). Here is an example `[manager]` section:

```
# MySQL Instance Manager options section
[manager]
default-mysqld-path = /usr/local/mysql/libexec/mysqld
socket=/tmp/manager.sock
pid-file=/tmp/manager.pid
password-file = /home/cps/.mysqlmanager.passwd
monitoring-interval = 2
port = 1999
bind-address = 192.168.1.5
```

Warning

The `[mysqld]` section name is deprecated and should not be used in a configuration file, instead `[mysqldN]` sections such as `[mysqld1]` should be used for specific instances.

Prior to MySQL 5.0.10, the MySQL Instance Manager read the same configuration files as the MySQL Server, including `/etc/my.cnf`, `~/.my.cnf`, etc. As of MySQL 5.0.10, the MySQL Instance Manager reads and manages the `/etc/my.cnf` file only on Unix. On Windows, MySQL Instance Manager reads the `my.ini` file in the directory where Instance Manager is installed. The default option file location can be changed with the `--defaults-file=file_name` option.

Instance sections specify options given to each instance at startup. These are mainly common MySQL server options, but there are some IM-specific options:

- `mysqld-path = path`

The pathname to the **mysqld** server binary.

- shutdown-delay = seconds

The number of seconds IM should wait for the instance to shut down. The default value is 35 seconds. After the delay expires, the IM assumes that the instance is hanging and attempts to terminate it. If you use InnoDB with large tables, you should increase this value.

- nonguarded

This option should be specified if you want to disable IM monitoring functionality for a certain instance.

Here are some sample instance sections:

```
[mysqld1]
mysqld-path=/usr/local/mysql/libexec/mysqld
socket=/tmp/mysql.sock
port=3307
server_id=1
skip-stack-trace
core-file
skip-bdb
log-bin
log-error
log=mylog
log-slow-queries

[mysqld2]
nonguarded
port=3308
server_id=2
mysqld-path= /home/cps/mysql/trees/mysql-5.0/sql/mysqld
socket      = /tmp/mysql.sock5
pid-file    = /tmp/hostname.pid5
datadir= /home/cps/mysql_data/data_dir1
language=/home/cps/mysql/trees/mysql-5.0/sql/share/english
log-bin
log=/tmp/fordel.log
```

5.5.5. Commands Recognized by the MySQL Instance Manager

Once you've set up a password file for the MySQL Instance Manager and the IM is running, you can connect to it. You can use the **mysql** client tool connect through a standard MySQL API:

```
mysql --port=2273 --host=mydomain.org --user=mysql -p
```

The following list of commands shows the MySQL Instance Manager currently accepts, with samples.

- `START INSTANCE instance_name`

This command attempts to start an instance.

```
mysql> START INSTANCE mysql4;  
Query OK, 0 rows affected (0,00 sec)
```

- `STOP INSTANCE instance_name`

This command attempts to stop an instance.

```
mysql> STOP INSTANCE mysql4;  
Query OK, 0 rows affected (0,00 sec)
```

- `SHOW INSTANCES`

Shows the names of all loaded instances.

```
mysql> SHOW INSTANCES;  
+-----+-----+  
| instance_name | status |  
+-----+-----+  
| mysql3       | offline |  
| mysql4       | online  |  
| mysql2       | offline |  
+-----+-----+  
3 rows in set (0,04 sec)
```

- `SHOW INSTANCE STATUS instance_name`

Shows the status and the version information for an instance.

```
mysql> SHOW INSTANCE STATUS mysql3;  
+-----+-----+-----+  
| instance_name | status | version |  
+-----+-----+-----+  
| mysql3       | online | unknown |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

- `SHOW INSTANCE OPTIONS instance_name`

Shows the options used by an instance.

```
mysql> SHOW INSTANCE OPTIONS mysqld3;
+-----+-----+
| option_name | value
+-----+-----+
| instance_name | mysqld3
| mysqld-path | /home/cps/mysql/trees/mysql-4.1/sql/mysqld
| port | 3309
| socket | /tmp/mysql.sock3
| pid-file | hostname.pid3
| datadir | /home/cps/mysql_data/data_dir1/
| language | /home/cps/mysql/trees/mysql-4.1/sql/share/engl
+-----+-----+
7 rows in set (0.01 sec)
```

- `SHOW instance_name LOG FILES`

The command lists all log files used by the instance. The result set contains the path to the log file and the log file size. If no log file path is specified in the configuration file (for example, `log=/var/mysql.log`), the Instance Manager tries to guess its placement. If the IM is unable to guess the logfile placement you should specify the log file location explicitly by using the appropriate log option in the instance section of the configuration file.

```
mysql> SHOW mysqld LOG FILES;
+-----+-----+-----+
| Logfile | Path | Filesize |
+-----+-----+-----+
| ERROR LOG | /home/cps/var/mysql/owlet.err | 9186 |
| GENERAL LOG | /home/cps/var/mysql/owlet.log | 471503 |
| SLOW LOG | /home/cps/var/mysql/owlet-slow.log | 4463 |
+-----+-----+-----+
3 rows in set (0.01 sec)
```

- `SHOW instance_name LOG {ERROR | SLOW | GENERAL} size[,offset_from_end]`

This command retrieves a portion of the specified log file. Because most users are interested in the latest log messages, the *size* parameter defines the number of bytes you would like to retrieve starting from the log end. You can retrieve data from the middle of the log file by specifying the

optional *offset_from_end* parameter. The following example retrieves 21 bytes of data, starting 23 bytes from the end of the log file and ending 2 bytes from the end of the log file:

```
mysql> SHOW mysqlld LOG GENERAL 21, 2;
+-----+
| Log                |
+-----+
| using password: YES |
+-----+
1 row in set (0.00 sec)
```

- `SET instance_name.option_name=option_value`

This command edits the specified instance's configuration file to change or add instance options. The IM assumes that the configuration file is located at `/etc/my.cnf`. You should check that the file exists and has appropriate permissions.

```
mysql> SET mysqlld2.port=3322;
Query OK, 0 rows affected (0.00 sec)
```

Changes made to the configuration file do not take effect until the MySQL server is restarted. In addition, these changes are not stored in the instance manager's local cache of instance settings until a `FLUSH INSTANCES` command is executed.

- `UNSET instance_name.option_name`

This command removes an option from an instance's configuration file.

```
mysql> UNSET mysqlld2.port;
Query OK, 0 rows affected (0.00 sec)
```

Changes made to the configuration file do not take effect until the MySQL server is restarted. In addition, these changes are not stored in the instance manager's local cache of instance settings until a `FLUSH INSTANCES` command is executed.

- `FLUSH INSTANCES`

This command forces IM to reread the configuration file and to refresh

internal structures. This command should be performed after editing the configuration file. The command does not restart instances.

```
mysql> FLUSH INSTANCES;  
Query OK, 0 rows affected (0.04 sec)
```

5.6. Installation-Related Programs

5.6.1. `mysql_fix_privilege_tables` — Upgrade MySQL System Tables

Some releases of MySQL introduce changes to the structure of the system tables in the `mysql` database to add new privileges or support new features. When you update to a new version of MySQL, you should update your system tables as well to make sure that their structure is up to date. Otherwise, there might be capabilities that you cannot take advantage of. First, make a backup of your `mysql` database, and then use the following procedure.

Note: As of MySQL 5.0.19, `mysql_fix_privilege_tables` is superseded by `mysql_upgrade`, which should be used instead. See [Section 5.6.2, “`mysql_upgrade` — Check Tables for MySQL Upgrade”](#).

On Unix or Unix-like systems, update the system tables by running the `mysql_fix_privilege_tables` script:

```
shell> mysql_fix_privilege_tables
```

You must run this script while the server is running. It attempts to connect to the server running on the local host as `root`. If your `root` account requires a password, indicate the password on the command line like this:

```
shell> mysql_fix_privilege_tables --password=root_password
```

The `mysql_fix_privilege_tables` script performs any actions necessary to convert your system tables to the current format. You might see some `Duplicate column name` warnings as it runs; you can ignore them.

After running the script, stop the server and restart it.

On Windows systems, MySQL distributions include a `mysql_fix_privilege_tables.sql` SQL script that you can run using the `mysql` client. For example, if your MySQL installation is located at `C:\Program Files\MySQL\MySQL Server 5.0`, the commands look like this:

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 5.0"
```

```
C:\> bin\mysql -u root -p mysql
mysql> SOURCE scripts/mysql_fix_privilege_tables.sql
```

The **mysql** command will prompt you for the root password; enter it when prompted.

If your installation is located in some other directory, adjust the pathnames appropriately.

As with the Unix procedure, you might see some Duplicate column name warnings as **mysql** processes the statements in the `mysql_fix_privilege_tables.sql` script; you can ignore them.

After running the script, stop the server and restart it.

5.6.2. **mysql_upgrade** — Check Tables for MySQL Upgrade

mysql_upgrade should be executed each time you upgrade MySQL. It checks all tables in all databases for incompatibilities with the current version of MySQL Server. If a table is found to have a possible incompatibility, it is checked. If any problems are found, the table is repaired. **mysql_upgrade** also upgrades the system tables so that you can take advantage of new privileges or capabilities that might have been added.

All checked and repaired tables are marked with the current MySQL version number. This ensures that next time you run **mysql_upgrade** with the same version of the server, it can tell whether there is any need to check or repair the table again.

mysql_upgrade also saves the MySQL version number in a file named `mysql_upgrade.info` in the data directory. This is used to quickly check if all tables have been checked for this release so that table-checking can be skipped. To ignore this file, use the `--force` option.

To check and repair tables and to upgrade the system tables, **mysql_upgrade** executes the following commands:

```
mysqlcheck --check-upgrade --all-databases --auto-repair
mysql_fix_privilege_tables
```

mysql_upgrade supersedes the older **mysql_fix_privilege_tables** script. In

MySQL 5.0.19, **mysql_upgrade** was added as a shell script and worked only for Unix systems. As of MySQL 5.0.23, **mysql_upgrade** is an executable binary and is available on all systems. On systems older than those supporting **mysql_upgrade**, you can execute the **mysqlcheck** command manually, and then upgrade your system tables as described in [Section 5.6.1, “mysql fix privilege tables — Upgrade MySQL System Tables”](#).

For details about what is checked, see the description of the FOR UPGRADE option of the CHECK TABLE statement (see [Section 13.5.2.3, “CHECK TABLE Syntax”](#)).

To use **mysql_upgrade**, make sure that the server is running, and then invoke it like this:

```
shell> mysql_upgrade [options]
```

mysql_upgrade reads options from the command line and from the [mysqld] and [mysql_upgrade] groups in option files. It supports the following options:

- --help

Display a short help message and exit.

- --basedir=path

The path to the MySQL installation directory.

- --datadir=path

The path to the data directory.

- --force

Force execution of **mysqlcheck** even if **mysql_upgrade** has already been executed for the current version of MySQL. (In other words, this option causes the `mysql_upgrade.info` file to be ignored.)

- --user=user_name, -u user_name

The MySQL username to use when connecting to the server. The default username is root.

- `--verbose`

Verbose mode. Print more information about what the program does.

Other options are passed to **mysqlcheck** and to **mysql_fix_privilege_tables**. For example, it might be necessary to specify the `--password[=password]` option.

5.7. General Security Issues

This section describes some general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see [Section 5.8, “The MySQL Access Privilege System”](#).

5.7.1. General Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, we emphasize the necessity of fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines whenever possible:

- **Do not ever give anyone (except MySQL root accounts) access to the user table in the mysql database!** This is critical.
- Learn the MySQL access privilege system. The GRANT and REVOKE statements are used for controlling access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

Checklist:

- Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL root user with full privileges! Review

the MySQL installation instructions, paying particular attention to the information about setting a root password. See [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

- Use the `SHOW GRANTS` statement to check which accounts have access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.
- Do not store any plain-text passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `MD5()`, `SHA1()`, or some other one-way hashing function and store the hash value.
- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “duag98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Mary had a little lamb” results in a password of “Mhall”). The password is easy to remember and type, but difficult to guess for someone who does not know the sentence.
- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

Checklist:

- Try to scan your ports from the Internet using a tool such as `nmap`. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. Another simple way to check whether or not your MySQL port is open is to try the following command from some remote machine, where `server_host` is the hostname or IP number of the host on which your MySQL server runs:

```
shell> telnet server_host 3306
```

If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open. If **telnet** hangs or the connection is

refused, the port is blocked, which is how you want it to be.

- Do not trust any data entered by users of your applications. They can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like “; DROP DATABASE mysql;”. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value 234, the user can enter the value `234 OR 1=1` to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotes around the numeric constants: `SELECT * FROM table WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing non-numeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is allowable to display any row in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

- Try to enter single and double quote marks (‘ ’ and ‘ ” ’) in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.
- Try to modify dynamic URLs by adding %22 (‘ ” ’), %23 (‘ # ’), and %27 (‘ ’ ’) to them.
- Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your

application should be safe against these and similar attacks.

- Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!
- Check the size of data before passing it to MySQL.
- Have your application connect to the database using a username different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.
- Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:
 - MySQL C API: Use the `mysql_real_escape_string()` API call.
 - MySQL++: Use the `escape` and `quote` modifiers for query streams.
 - PHP: Use the `mysql_real_escape_string()` function (available as of PHP 4.3.0, prior to that PHP version use `mysql_escape_string()`, and prior to PHP 4.0.3, use `addslashes()`). Note that only `mysql_real_escape_string()` is character set-aware; the other functions can be “bypassed” when using (invalid) multi-byte character sets. In PHP 5, you can use the `mysqli` extension, which supports the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders.
 - Perl DBI: Use placeholders or the `quote()` method.
 - Ruby DBI: Use placeholders or the `quote()` method.
 - Java JDBC: Use a `PreparedStatement` object and placeholders.

Other programming interfaces might have similar capabilities.

- Do not transmit plain (unencrypted) data over the Internet. This information

is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections as of version 4.0. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.

- Learn to use the **tcpdump** and **strings** utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

(This works under Linux and should work with small modifications under other systems.) Warning: If you do not see plaintext data, this doesn't always mean that the information actually is encrypted. If you need high security, you should consult with a security expert.

5.7.2. Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted in clear text over the connection. Password handling during the client connection sequence was upgraded in MySQL 4.1.1 to be very secure. If you are still using pre-4.1.1-style passwords, the encryption algorithm is not as strong as the newer algorithm. With some effort, a clever attacker who can sniff the traffic between the client and the server can crack the password. (See [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#), for a discussion of the different password handling methods.)

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure. See [Section 5.9.7, “Using Secure Connections”](#). Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at <http://www.openssh.org/>, and a commercial SSH client at <http://www.ssh.com/>.

To make a MySQL system secure, you should strongly consider the following

suggestions:

- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any username to the client program. For example, anyone can use the **mysql** program to connect as any other person simply by invoking it as `mysql -u other_user db_name` if *other_user* has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

For a discussion of methods for setting passwords, see [Section 5.9.5, “Assigning Account Passwords”](#).

- Never run the MySQL server as the Unix root user. This is extremely dangerous, because any user with the FILE privilege is able to cause the server to create files as root (for example, `~root/.bashrc`). To prevent this, **mysqld** refuses to run as root unless that is specified explicitly using the `--user=root` option.

mysqld can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start **mysqld** as a different Unix user, add a `user` option that specifies the username in the `[mysqld]` group of the `my.cnf` option file where you specify server options. For example:

```
[mysqld]
user=mysql
```

This causes the server to start as the designated user whether you start it manually or by using **mysqld_safe** or **mysql.server**. For more details, see [Section 5.7.5, “How to Run MySQL as a Normal User”](#).

Running **mysqld** as a Unix user other than root does not mean that you need to change the root username in the user table. *Usernames for MySQL accounts have nothing to do with usernames for Unix accounts.*

- Do not allow the use of symlinks to tables. (This capability can be disabled with the `--skip-symbolic-links` option.) This is especially important if you run **mysqld** as root, because anyone that has write access to the

server's data directory then could delete any file in the system! See [Section 7.6.1.2, “Using Symbolic Links for Tables on Unix”](#).

- Make sure that the only Unix user with read or write privileges in the database directories is the user that **mysqld** runs as.
- Do not grant the `PROCESS` or `SUPER` privilege to non-administrative users. The output of **mysqladmin processlist** and `SHOW PROCESSLIST` shows the text of any statements currently being executed, so any user who is allowed to see the server process list might be able to see statements issued by other users such as `UPDATE user SET password=PASSWORD('not_secure')`.

mysqld reserves an extra connection for users who have the `SUPER` privilege, so that a MySQL root user can log in and check server activity even if all normal connections are in use.

The `SUPER` privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not grant the `FILE` privilege to non-administrative users. Any user that has this privilege can write a file anywhere in the filesystem with the privileges of the **mysqld** daemon. To make this a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

The `FILE` privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

- If you do not trust your DNS, you should use IP numbers rather than hostnames in the grant tables. In any case, you should be very careful about creating grant table entries using hostname values that contain wildcards.
- If you want to restrict the number of connections allowed to a single account, you can do so by setting the `max_user_connections` variable in **mysqld**. The `GRANT` statement also supports resource control options for limiting the extent of server use allowed to an account. See

[Section 13.5.1.3, “GRANT Syntax”](#).

- `--ssl*`

Options that begin with `--ssl` specify whether to allow clients to connect via SSL and indicate where to find SSL keys and certificates. See [Section 5.9.7.3, “SSL Command Options”](#).

5.7.3. Security-Related `mysqld` Options

The following `mysqld` options affect security:

- `--allow-suspicious-udfs`

This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. For MySQL 5.0, this option was added in MySQL 5.0.3. See [Section 24.2.4.6, “User-Defined Function Security Precautions”](#).

- `--local-infile[={0|1}]`

If you start the server with `--local-infile=0`, clients cannot use `LOCAL` in `LOAD DATA` statements. See [Section 5.7.4, “Security Issues with `LOAD DATA LOCAL`”](#).

- `--old-passwords`

Force the server to generate short (pre-4.1) password hashes for new passwords. This is useful for compatibility when the server must support older client programs. See [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#).

- `--safe-show-database` (*OBSOLETE*)

In previous versions of MySQL, this option caused the `SHOW DATABASES` statement to display the names of only those databases for which the user had some kind of privilege. In MySQL 5.0, this option is no longer

available as this is now the default behavior, and there is a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See [Section 13.5.1.3, “GRANT Syntax”](#).

- `--safe-user-create`

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` privilege for the `mysql.user` table. If you want a user to have the ability to create new users that have those privileges that the user has right to grant, you should grant the user the following privilege:

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

This ensures that the user cannot change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--secure-auth`

Disallow authentication for accounts that have old (pre-4.1) passwords.

The **mysql** client also has a `--secure-auth` option, which prevents connections to a server if the server requires a password in old format for the client account.

- `--skip-grant-tables`

This option causes the server not to use the privilege system at all. This gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing **mysqladmin flush-privileges** or **mysqladmin reload** command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement. This option also suppresses loading of user-defined functions (UDFs).

- `--skip-name-resolve`

Hostnames are not resolved. All `Host` column values in the grant tables must be IP numbers or `localhost`.

- `--skip-networking`

Do not allow TCP/IP connections over the network. All connections to **mysqld** must be made via Unix socket files.

- `--skip-show-database`

With this option, the `SHOW DATABASES` statement is allowed only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. Without this option, `SHOW DATABASES` is allowed to all users, but displays each database name only if the user has the `SHOW DATABASES` privilege or some privilege for the database. Note that any global privilege is a privilege for the database.

5.7.4. Security Issues with `LOAD DATA LOCAL`

The `LOAD DATA` statement can load a file that is located on the server host, or it can load a file that is located on the client host when the `LOCAL` keyword is specified.

There are two potential security issues with supporting the `LOCAL` version of `LOAD DATA` statements:

- The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD DATA` statement. Such a server could access any file on the client host to which the client user has read access.
- In a Web environment where the clients are connecting from a Web server, a user could use `LOAD DATA LOCAL` to read any files that the Web server process has read access to (assuming that a user could run any command against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not the remote program being run by the user who connects to the Web server.

To deal with these problems, we changed how `LOAD DATA LOCAL` is handled as of MySQL 3.23.49 and MySQL 4.0.2 (4.0.13 on Windows):

- By default, all MySQL clients and libraries in binary distributions are compiled with the `--enable-local-infile` option, to be compatible with

MySQL 3.23.48 and before.

- If you build MySQL from source but do not invoke **configure** with the `--enable-local-infile` option, `LOAD DATA LOCAL` cannot be used by any client unless it is written explicitly to invoke `mysql_options(... MYSQL_OPT_LOCAL_INFILE, 0)`. See [Section 22.2.3.48, “mysql_options\(\)”](#).
- You can disable all `LOAD DATA LOCAL` commands from the server side by starting **mysqld** with the `--local-infile=0` option.
- For the **mysql** command-line client, `LOAD DATA LOCAL` can be enabled by specifying the `--local-infile[=1]` option, or disabled with the `--local-infile=0` option. Similarly, for **mysqlimport**, the `--local` or `-L` option enables local data file loading. In any case, successful use of a local loading operation requires that the server is enabled to allow it.
- If you use `LOAD DATA LOCAL` in Perl scripts or other programs that read the `[client]` group from option files, you can add the `local-infile=1` option to that group. However, to keep this from causing problems for programs that do not understand `local-infile`, specify it using the `loose-` prefix:

```
[client]
loose-local-infile=1
```

- If `LOAD DATA LOCAL INFILE` is disabled, either in the server or the client, a client that attempts to issue such a statement receives the following error message:

```
ERROR 1148: The used command is not allowed with this MySQL vers
```

5.7.5. How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account.

On Unix, the MySQL server **mysqld** can be started and run by any user. However, you should avoid running the server as the Unix root user for security reasons. To change **mysqld** to run as a normal unprivileged Unix user *user_name*, you must do the following:

1. Stop the server if it's running (use **mysqladmin shutdown**).
2. Change the database directories and files so that *user_name* has privileges to read and write files in them (you might need to do this as the Unix root user):

```
shell> chown -R user_name /path/to/mysql/datadir
```

If you do not do this, the server will not be able to access databases or tables when it runs as *user_name*.

If directories or files within the MySQL data directory are symbolic links, you'll also need to follow those links and change the directories and files they point to. `chown -R` might not follow symbolic links for you.

3. Start the server as user *user_name*. If you are using MySQL 3.22 or later, another alternative is to start **mysqld** as the Unix root user and use the `--user=user_name` option. **mysqld** starts up, then switches to run as the Unix user *user_name* before accepting any connections.
4. To start the server as the given user automatically at system startup time, specify the username by adding a `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

```
[mysqld]
user=user_name
```

If your Unix machine itself isn't secured, you should assign passwords to the MySQL root accounts in the grant tables. Otherwise, any user with a login account on that machine can run the **mysql** client with a `--user=root` option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See [Section 2.10, "Post-Installation Setup and Testing"](#).

5.8. The MySQL Access Privilege System

MySQL has an advanced but non-standard security and privilege system. The following discussion describes how it works.

5.8.1. What the Privilege System Does

The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.

Additional functionality includes the ability to have anonymous users and to grant privileges for MySQL-specific functions such as `LOAD DATA INFILE` and administrative operations.

5.8.2. How the Privilege System Works

The MySQL privilege system ensures that all users may perform only the operations allowed to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the username you specify*. When you issue requests after connecting, the system grants privileges according to your identity and *what you want to do*.

MySQL considers both your hostname and username in identifying you because there is little reason to assume that a given username belongs to the same person everywhere on the Internet. For example, the user `joe` who connects from `office.example.com` need not be the same person as the user `joe` who connects from `home.example.com`. MySQL handles this by allowing you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by `joe` from `office.example.com`, and a different set of privileges for connections by `joe` from `home.example.com`.

MySQL access control involves two stages when you run a client program that connects to the server:

- Stage 1: The server checks whether it should allow you to connect.
- Stage 2: Assuming that you can connect, the server checks each statement

you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the `SELECT` privilege for the table or the `DROP` privilege for the database.

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. See [Section 5.8.7, “When Privilege Changes Take Effect”](#), for details.

The server stores privilege information in the grant tables of the `mysql` database (that is, in the database named `mysql`). The MySQL server reads the contents of these tables into memory when it starts and re-reads them under the circumstances indicated in [Section 5.8.7, “When Privilege Changes Take Effect”](#). Access-control decisions are based on the in-memory copies of the grant tables.

Normally, you manipulate the contents of the grant tables indirectly by using statements such as `GRANT` and `REVOKE` to set up accounts and control the privileges available to each one. See [Section 13.5.1, “Account Management Statements”](#). The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients.

The server uses the `user`, `db`, and `host` tables in the `mysql` database at both stages of access control. The columns in the `user` and `db` tables are shown here. The `host` table is similar to the `db` table but has a specialized use as described in [Section 5.8.6, “Access Control, Stage 2: Request Verification”](#).

Table Name	user	db
Scope columns	Host	Host
	User	Db
	Password	User
Privilege columns	Select_priv	Select_priv
	Insert_priv	Insert_priv
	Update_priv	Update_priv
	Delete_priv	Delete_priv
	Index_priv	Index_priv

	Alter_priv	Alter_priv
	Create_priv	Create_priv
	Drop_priv	Drop_priv
	Grant_priv	Grant_priv
	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv
	Alter_routine_priv	Alter_routine_priv
	Execute_priv	Execute_priv
	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv
	References_priv	References_priv
	Reload_priv	
	Shutdown_priv	
	Process_priv	
	File_priv	
	Show_db_priv	
	Super_priv	
	Repl_slave_priv	
	Repl_client_priv	
Security columns	ssl_type	
	ssl_cipher	
	x509_issuer	
	x509_subject	
Resource control columns	max_questions	
	max_updates	
	max_connections	
	max_user_connections	

Execute_priv was present in MySQL 5.0.0, but did not become operational until MySQL 5.0.3.

The `Create_view_priv` and `Show_view_priv` columns were added in MySQL 5.0.1.

The `Create_routine_priv`, `Alter_routine_priv`, and `max_user_connections` columns were added in MySQL 5.0.3.

During the second stage of access control, the server performs request verification to make sure that each client has sufficient privileges for each request that it issues. In addition to the `user`, `db`, and `host` grant tables, the server may also consult the `tables_priv` and `columns_priv` tables for requests that involve tables. The `tables_priv` and `columns_priv` tables provide finer privilege control at the table and column levels. They have the following columns:

Table Name	tables_priv	columns_priv
Scope columns	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Privilege columns	Table_priv	Column_priv
	Column_priv	
Other columns	Timestamp	Timestamp
	Grantor	

The `Timestamp` and `Grantor` columns currently are unused and are discussed no further here.

For verification of requests that involve stored routines, the server may consult the `procs_priv` table. This table has the following columns:

Table Name	procs_priv
Scope columns	Host
	Db
	User
	Routine_name

	Routine_type
Privilege columns	Proc_priv
Other columns	Timestamp
	Grantor

The `procs_priv` table exists as of MySQL 5.0.3. The `Routine_type` column was added in MySQL 5.0.6. It is an `ENUM` column with values of `'FUNCTION'` or `'PROCEDURE'` to indicate the type of routine the row refers to. This column allows privileges to be granted separately for a function and a procedure with the same name.

The `Timestamp` and `Grantor` columns currently are unused and are discussed no further here.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row (entry) in the tables; that is, the context in which the row applies. For example, a user table row with `Host` and `User` values of `'thomas.loc.gov'` and `'bob'` would be used for authenticating connections made to the server from the host `thomas.loc.gov` by a client that specifies a username of `bob`. Similarly, a `db` table row with `Host`, `User`, and `Db` column values of `'thomas.loc.gov'`, `'bob'` and `'reports'` would be used when `bob` connects from the host `thomas.loc.gov` to access the `reports` database. The `tables_priv` and `columns_priv` tables contain scope columns indicating tables or table/column combinations to which each row applies. The `procs_priv` scope columns indicate the stored routine to which each row applies.
- Privilege columns indicate which privileges are granted by a table row; that is, what operations can be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. [Section 5.8.6, “Access Control, Stage 2: Request Verification”](#), describes the rules that are used to do this.

Scope columns contain strings. They are declared as shown here; the default value for each is the empty string:

Column Name	Type
Host	CHAR(60)
User	CHAR(16)
Password	CHAR(16)
Db	CHAR(64)
Table_name	CHAR(64)
Column_name	CHAR(64)
Routine_name	CHAR(64)

For access-checking purposes, comparisons of Host values are case-insensitive. User, Password, Db, and Table_name values are case sensitive. Column_name and Routine_name values are case insensitive.

In the user, db, and host tables, each privilege is listed in a separate column that is declared as ENUM('N', 'Y') DEFAULT 'N'. In other words, each privilege can be disabled or enabled, with the default being disabled.

In the tables_priv, columns_priv, and procs_priv tables, the privilege columns are declared as SET columns. Values in these columns can contain any combination of the privileges controlled by the table:

Table Name	Column Name	Possible Set Elements
tables_priv	Table_priv	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view'
tables_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
columns_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
procs_priv	Proc_priv	'Execute', 'Alter Routine', 'Grant'

Briefly, the server uses the grant tables in the following manner:

- The user table scope columns determine whether to reject or allow incoming connections. For allowed connections, any privileges granted in the user table indicate the user's global (superuser) privileges. Any privilege granted in this table applies to *all* databases on the server.

Note: Because any global privilege is considered a privilege for all databases, any global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`.

- The `db` table scope columns determine which users can access which databases from which hosts. The privilege columns determine which operations are allowed. A privilege granted at the database level applies to the database and to all its tables.
- The `host` table is used in conjunction with the `db` table when you want a given `db` table row to apply to several hosts. For example, if you want a user to be able to use a database from several hosts in your network, leave the `Host` value empty in the user's `db` table row, then populate the `host` table with a row for each of those hosts. This mechanism is described more detail in [Section 5.8.6, “Access Control, Stage 2: Request Verification”](#).

Note: The `host` table must be modified directly with statements such as `INSERT`, `UPDATE`, and `DELETE`. It is not affected by statements such as `GRANT` and `REVOKE` that modify the grant tables indirectly. Most MySQL installations need not use this table at all.

- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.
- The `procs_priv` table applies to stored routines. A privilege granted at the routine level applies only to a single routine.

Administrative privileges (such as `RELOAD` or `SHUTDOWN`) are specified only in the `user` table. The reason for this is that administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. In fact, to determine whether you can perform an administrative operation, the server need consult only the `user` table.

The `FILE` privilege also is specified only in the `user` table. It is not an administrative privilege as such, but your ability to read or write files on the server host is independent of the database you are accessing.

The **mysqld** server reads the contents of the grant tables into memory when it starts. You can tell it to re-read the tables by issuing a `FLUSH PRIVILEGES` statement or executing a **mysqladmin flush-privileges** or **mysqladmin reload** command. Changes to the grant tables take effect as indicated in [Section 5.8.7, “When Privilege Changes Take Effect”](#).

When you modify the contents of the grant tables, it is a good idea to make sure that your changes set up privileges the way you want. To check the privileges for a given account, use the `SHOW GRANTS` statement. (See [Section 13.5.4.12, “SHOW GRANTS Syntax”](#).) For example, to determine the privileges that are granted to an account with Host and User values of `pc84.example.com` and `bob`, issue this statement:

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

For additional help in diagnosing privilege-related problems, see [Section 5.8.8, “Causes of Access denied Errors”](#). For general advice on security issues, see [Section 5.7, “General Security Issues”](#).

5.8.3. Privileges Provided by MySQL

Information about account privileges is stored in the `user`, `db`, `host`, `tables_priv`, `columns_priv`, and `procs_priv` tables in the `mysql` database. The MySQL server reads the contents of these tables into memory when it starts and re-reads them under the circumstances indicated in [Section 5.8.7, “When Privilege Changes Take Effect”](#). Access-control decisions are based on the in-memory copies of the grant tables.

The names used in the `GRANT` and `REVOKE` statements to refer to privileges are shown in the following table, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies. Further information about the meaning of each privilege may be found at [Section 13.5.1.3, “GRANT Syntax”](#).

Privilege	Column	Context
CREATE	Create_priv	databases, tables, or indexes
DROP	Drop_priv	databases or tables
GRANT OPTION	Grant_priv	databases, tables, or stored

		routines
REFERENCES	References_priv	databases or tables
ALTER	Alter_priv	tables
DELETE	Delete_priv	tables
INDEX	Index_priv	tables
INSERT	Insert_priv	tables
SELECT	Select_priv	tables
UPDATE	Update_priv	tables
CREATE VIEW	Create_view_priv	views
SHOW VIEW	Show_view_priv	views
ALTER ROUTINE	Alter_routine_priv	stored routines
CREATE ROUTINE	Create_routine_priv	stored routines
EXECUTE	Execute_priv	stored routines
FILE	File_priv	file access on server host
CREATE TEMPORARY TABLES	Create_tmp_table_priv	server administration
LOCK TABLES	Lock_tables_priv	server administration
CREATE USER	Create_user_priv	server administration
PROCESS	Process_priv	server administration
RELOAD	Reload_priv	server administration
REPLICATION CLIENT	Repl_client_priv	server administration
REPLICATION SLAVE	Repl_slave_priv	server administration
SHOW DATABASES	Show_db_priv	server administration
SHUTDOWN	Shutdown_priv	server administration
SUPER	Super_priv	server administration

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

CREATE VIEW and SHOW VIEW were added in MySQL 5.0.1. CREATE USER, CREATE

ROUTINE, and ALTER ROUTINE were added in MySQL 5.0.3. Although EXECUTE was present in MySQL 5.0.0, it did not become operational until MySQL 5.0.3.

To create or alter stored routines if binary logging is enabled, you may also need the SUPER privilege, as described in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

The CREATE and DROP privileges allow you to create new databases and tables, or to drop (remove) existing databases and tables. *If you grant the DROP privilege for the mysql database to a user, that user can drop the database in which the MySQL access privileges are stored.*

The SELECT, INSERT, UPDATE, and DELETE privileges allow you to perform operations on rows in existing tables in a database. INSERT is also required for the ANALYZE TABLE, OPTIMIZE TABLE, and REPAIR TABLE table-maintenance statements.

SELECT statements require the SELECT privilege only if they actually retrieve rows from a table. Some SELECT statements do not access tables and can be executed without permission for any database. For example, you can use the **mysql** client as a simple calculator to evaluate expressions that make no reference to tables:

```
SELECT 1+1;  
SELECT PI()*2;
```

The INDEX privilege enables you to create or drop (remove) indexes. INDEX applies to existing tables. If you have the CREATE privilege for a table, you can include index definitions in the CREATE TABLE statement.

The ALTER privilege enables you to use ALTER TABLE to change the structure of or rename tables.

The CREATE ROUTINE privilege is needed for creating stored routines (functions and procedures). ALTER ROUTINE privilege is needed for altering or dropping stored routines, and EXECUTE is needed for executing stored routines.

The GRANT privilege enables you to give to other users those privileges that you yourself possess. It can be used for databases, tables, and stored routines.

The FILE privilege gives you permission to read and write files on the server host using the LOAD DATA INFILE and SELECT ... INTO OUTFILE statements. A user who has the FILE privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.) The FILE privilege also enables the user to create new files in any directory where the MySQL server has write access. As a security measure, the server will not overwrite existing files.

The remaining privileges are used for administrative operations. Many of them can be performed by using the **mysqladmin** program or by issuing SQL statements. The following table shows which **mysqladmin** commands each administrative privilege enables you to execute:

Privilege	Commands Permitted to Privilege Holders
RELOAD	flush-hosts, flush-logs, flush-privileges, flush-status, flush-tables, flush-threads, refresh, reload
SHUTDOWN	shutdown
PROCESS	processlist
SUPER	kill

The reload command tells the server to re-read the grant tables into memory. flush-privileges is a synonym for reload. The refresh command closes and reopens the log files and flushes all tables. The other flush-xxx commands perform functions similar to refresh, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, flush-logs is a better choice than refresh.

The shutdown command shuts down the server. There is no corresponding SQL statement.

The processlist command displays information about the threads executing within the server (that is, information about the statements being executed by clients). The kill command terminates server threads. You can always display or kill your own threads, but you need the PROCESS privilege to display threads initiated by other users and the SUPER privilege to kill them. See [Section 13.5.5.3, “KILL Syntax”](#).

The CREATE TEMPORARY TABLES privilege enables the use of the keyword

TEMPORARY in CREATE TABLE statements.

The LOCK TABLES privilege enables the use of explicit LOCK TABLES statements to lock tables for which you have the SELECT privilege. This includes the use of write locks, which prevents anyone else from reading the locked table.

The REPLICATION CLIENT privilege enables the use of SHOW MASTER STATUS and SHOW SLAVE STATUS.

The REPLICATION SLAVE privilege should be granted to accounts that are used by slave servers to connect to the current server as their master. Without this privilege, the slave cannot request updates that have been made to databases on the master server.

The SHOW DATABASES privilege allows the account to see database names by issuing the SHOW DATABASE statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the --skip-show-database option. Note that *any* global privilege is a privilege for the database.

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the FILE and administrative privileges:

- The FILE privilege can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using SELECT to transfer its contents to the client host.
- The GRANT privilege enables users to give their privileges to other users. Two users that have different privileges and with the GRANT privilege are able to combine privileges.
- The ALTER privilege may be used to subvert the privilege system by renaming tables.
- The SHUTDOWN privilege can be abused to deny service to other users entirely by terminating the server.
- The PROCESS privilege can be used to view the plain text of currently

executing statements, including statements that set or change passwords.

- The `SUPER` privilege can be used to terminate other clients or change how the server operates.
- Privileges granted for the `mysql` database itself can be used to change passwords and other access privilege information. Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the `user` table `Password` column can change an account's password, and then connect to the MySQL server using that account.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

5.8.4. Connecting to the MySQL Server

MySQL client programs generally expect you to specify certain connection parameters when you want to access a MySQL server:

- The name of the host where the MySQL server is running
- Your username
- Your password

For example, the **mysql** client can be started as follows from a command-line prompt (indicated here by `shell>`):

```
shell> mysql -h host_name -u user_name -pyour_pass
```

Alternative forms of the `-h`, `-u`, and `-p` options are `--host=host_name`, `--`

user=user_name, and --password=your_pass. Note that there is *no space* between -p or --password= and the password following it.

If you use a -p or --password option but do not specify the password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line. Any user on your system may be able to see a password specified on the command line by executing a command such as **ps auxww**. See [Section 5.9.6, “Keeping Your Password Secure”](#).

MySQL client programs use default values for any connection parameter option that you do not specify:

- The default hostname is localhost.
- The default username is ODBC on Windows and your Unix login name on Unix.
- No password is supplied if neither -p nor --password is given.

Thus, for a Unix user with a login name of joe, all of the following commands are equivalent:

```
shell> mysql -h localhost -u joe
shell> mysql -h localhost
shell> mysql -u joe
shell> mysql
```

Other MySQL clients behave similarly.

You can specify different default values to be used when you make a connection so that you need not enter them on the command line each time you invoke a client program. This can be done in a couple of ways:

- You can specify connection parameters in the [client] section of an option file. The relevant section of the file might look like this:

```
[client]
host=host_name
user=user_name
password=your_pass
```

[Section 4.3.2, “Using Option Files”](#), discusses option files further.

- You can specify some connection parameters using environment variables. The host can be specified for **mysql** using `MYSQL_HOST`. The MySQL username can be specified using `USER` (this is for Windows and NetWare only). The password can be specified using `MYSQL_PWD`, although this is insecure; see [Section 5.9.6, “Keeping Your Password Secure”](#). For a list of variables, see [Appendix F, *Environment Variables*](#).

5.8.5. Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password. If not, the server denies access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL username

Identity checking is performed using the three user table scope columns (Host, User, and Password). The server accepts the connection only if the Host and User columns in some user table row match the client hostname and username and the client supplies the password specified in that row.

Host values in the user table may be specified as follows:

- A Host value may be a hostname or an IP number, or 'localhost' to indicate the local host.
- You can use the wildcard characters '%' and '_' in Host column values. These have the same meaning as for pattern-matching operations performed with the LIKE operator. For example, a Host value of '%' matches any hostname, whereas a value of '%.mysql.com' matches any host in the mysql.com domain.
- For Host values specified as IP numbers, you can specify a netmask

indicating how many address bits to use for the network number. For example:

```
GRANT ALL PRIVILEGES ON db.* TO david@'192.58.197.0/255.255.255.
```

This allows david to connect from any client host having an IP number `client_ip` for which the following condition is true:

```
client_ip & netmask = host_ip
```

That is, for the GRANT statement just shown:

```
client_ip & 255.255.255.0 = 192.58.197.0
```

IP numbers that satisfy this condition and can connect to the MySQL server are those in the range from 192.58.197.0 to 192.58.197.255.

Note: The netmask can only be used to tell the server to use 8, 16, 24, or 32 bits of the address. Examples:

- 192.0.0.0/255.0.0.0: anything on the 192 class A network
- 192.168.0.0/255.255.0.0: anything on the 192.168 class B network
- 192.168.1.0/255.255.255.0: anything on the 192.168.1 class C network
- 192.168.1.1: only this specific IP

The following netmask (28 bits) will not work:

```
192.168.0.1/255.255.255.240
```

- A blank Host value in a db table row means that its privileges should be combined with those in the row in the host table that matches the client hostname. The privileges are combined using an AND (intersection) operation, not OR (union). [Section 5.8.6, “Access Control, Stage 2: Request Verification”](#), discusses use of the host table further.

A blank Host value in the other grant tables is the same as '%'.

Because you can use IP wildcard values in the Host column (for example,

'144.155.166.%' to match every host on a subnet), someone could try to exploit this capability by naming a host 144.155.166.somewhere.com. To foil such attempts, MySQL disallows matching on hostnames that start with digits and a dot. Thus, if you have a host named something like 1.2.foo.com, its name never matches the Host column of the grant tables. An IP wildcard value can match only IP numbers, not hostnames.

In the User column, wildcard characters are not allowed, but you can specify a blank value, which matches any name. If the user table row that matches an incoming connection has a blank username, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank username is used for all further access checking for the duration of the connection (that is, during Stage 2).

The Password column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password.

Non-blank Password values in the user table represent encrypted passwords. MySQL does not store passwords in plaintext form for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the PASSWORD() function). The encrypted password then is used during the connection process when checking whether the password is correct. (This is done without the encrypted password ever traveling over the connection.) From MySQL's point of view, the encrypted password is the *real* password, so you should never give anyone access to it. In particular, *do not give non-administrative users read access to tables in the mysql database.*

MySQL 5.0 employs the stronger authentication method (first implemented in MySQL 4.1) that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the mysql database is captured. [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#), discusses password encryption further.

The following table shows how various combinations of Host and User values in the user table apply to incoming connections.

Host Value	User Value	Allowable Connections
------------	------------	-----------------------

'thomas.loc.gov'	'fred'	fred, connecting from thomas.loc.gov
'thomas.loc.gov'	' '	Any user, connecting from thomas.loc.gov
'%'	'fred'	fred, connecting from any host
'%'	' '	Any user, connecting from any host
'%.loc.gov'	'fred'	fred, connecting from any host in the loc.gov domain
'x.y.%'	'fred'	fred, connecting from x.y.net, x.y.com, x.y.edu, and so on (this is probably not useful)
'144.155.166.177'	'fred'	fred, connecting from the host with IP address 144.155.166.177
'144.155.166.%'	'fred'	fred, connecting from any host in the 144.155.166 class C subnet
'144.155.166.0/255.255.255.0'	'fred'	Same as previous example

It is possible for the client hostname and username of an incoming connection to match more than one row in the user table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from thomas.loc.gov by fred.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the user table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client hostname and username.

To see how this works, suppose that the user table looks like this:

```
+-----+-----+
| Host      | User      | ...
+-----+-----+
```

%	root	...
%	jeffrey	...
localhost	root	...
localhost		...

When the server reads the table into memory, it orders the rows with the most-specific Host values first. Literal hostnames and IP numbers are the most specific. The pattern '%' means “any host” and is least specific. Rows with the same Host value are ordered with the most-specific User values first (a blank User value means “any user” and is least specific). For the user table just shown, the result after sorting looks like this:

Host	User	...
localhost	root	...
localhost		...
%	jeffrey	...
%	root	...

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from localhost by jeffrey, two of the rows from the table match: the one with Host and User values of 'localhost' and '', and the one with values of '%' and 'jeffrey'. The 'localhost' row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the user table looks like this:

Host	User	...
%	jeffrey	...
thomas.loc.gov		...

The sorted table looks like this:

Host	User	...
thomas.loc.gov		...
%	jeffrey	...

A connection by jeffrey from thomas.loc.gov is matched by the first row, whereas a connection by jeffrey from whitehouse.gov is matched by the second.

It is a common misconception to think that, for a given username, all rows that explicitly name that user are used first when the server attempts to find a match for the connection. This is simply not true. The previous example illustrates this, where a connection from thomas.loc.gov by jeffrey is first matched not by the row containing 'jeffrey' as the User column value, but by the row with no username. As a result, jeffrey is authenticated as an anonymous user, even though he specified a username when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the CURRENT_USER() function. (See [Section 12.9.3, “Information Functions”](#).) It returns a value in user_name@host_name format that indicates the User and Host values from the matching user table row. Suppose that jeffrey connects and issues the following query:

```
mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| @localhost     |
+-----+
```

The result shown here indicates that the matching user table row had a blank User column value. In other words, the server is treating jeffrey as an anonymous user.

Another thing you can do to diagnose authentication problems is to print out the user table and sort it by hand to see where the first match is being made.

5.8.6. Access Control, Stage 2: Request Verification

After you establish a connection, the server enters Stage 2 of access control. For each request that you issue via that connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come

into play. These privileges can come from any of the user, db, host, tables_priv, columns_priv, or procs_priv tables. (You may find it helpful to refer to [Section 5.8.2, “How the Privilege System Works”](#), which lists the columns present in each of the grant tables.)

The user table grants privileges that are assigned to you on a global basis and that apply no matter what the default database is. For example, if the user table grants you the DELETE privilege, you can delete rows from any table in any database on the server host! In other words, user table privileges are superuser privileges. It is wise to grant privileges in the user table only to superusers such as database administrators. For other users, you should leave all privileges in the user table set to 'N' and grant privileges at more specific levels only. You can grant privileges for particular databases, tables, columns, or routines.

The db and host tables grant database-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters '%' and '_' can be used in the Host and Db columns of either table. These have the same meaning as for pattern-matching operations performed with the LIKE operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include the underscore character ('_') as part of a database name, specify it as '_' in the GRANT statement.
- A '%' Host value in the db table means “any host.” A blank Host value in the db table means “consult the host table for further information” (a process that is described later in this section).
- A '%' or blank Host value in the host table means “any host.”
- A '%' or blank Db value in either table means “any database.”
- A blank User value in either table matches the anonymous user.

The server reads the db and host tables into memory and sorts them at the same time that it reads the user table. The server sorts the db table based on the Host, Db, and User scope columns, and sorts the host table based on the Host and Db scope columns. As with the user table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching entries, it uses the first match that it finds.

The `tables_priv`, `columns_priv`, and `procs_priv` tables grant table-specific, column-specific, and routine-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters ‘%’ and ‘_’ can be used in the `Host` column. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator.
- A ‘%’ or blank `Host` value means “any host.”
- The `Db`, `Table_name`, and `Column_name` columns cannot contain wildcards or be blank.

The server sorts the `tables_priv`, `columns_priv`, and `procs_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as `SHUTDOWN` or `RELOAD`, the server checks only the user table row because that is the only table that specifies administrative privileges. The server grants access if the row allows the requested operation and denies access otherwise. For example, if you want to execute **mysqladmin shutdown** but your user table row doesn't grant the `SHUTDOWN` privilege to you, the server denies access without even checking the `db` or `host` tables. (They contain no `Shutdown_priv` column, so there is no need to do so.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user's global (superuser) privileges by looking in the user table row. If the row allows the requested operation, access is granted. If the global privileges in the user table are insufficient, the server determines the user's database-specific privileges by checking the `db` and `host` tables:

1. The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns. The `Host` and `User` columns are matched to the connecting user's hostname and MySQL username. The `Db` column is matched to the database that the user wants to access. If there is no row for the `Host` and `User`, access is denied.
2. If there is a matching `db` table row and its `Host` column is not blank, that

row defines the user's database-specific privileges.

3. If the matching db table row's `Host` column is blank, it signifies that the host table enumerates which hosts should be allowed access to the database. In this case, a further lookup is done in the host table to find a match on the `Host` and `Db` columns. If no host table row matches, access is denied. If there is a match, the user's database-specific privileges are computed as the intersection (*not* the union!) of the privileges in the db and host table entries; that is, the privileges that are 'Y' in both entries. (This way you can grant general privileges in the db table row and then selectively restrict them on a host-by-host basis using the host table entries.)

After determining the database-specific privileges granted by the db and host table entries, the server adds them to the global privileges granted by the user table. If the result allows the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the `tables_priv` and `columns_priv` tables, adds those to the user's privileges, and allows or denies access based on the result. For stored routine operations, the server uses the `procs_priv` table rather than `tables_priv` and `columns_priv`.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

It may not be apparent why, if the global user row privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT INTO ... SELECT` statement, you need both the `INSERT` and the `SELECT` privileges. Your privileges might be such that the user table row grants one privilege and the db table row grants the other. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either table by itself; the privileges granted by the entries in both tables must be combined.

The host table is not affected by the GRANT or REVOKE statements, so it is unused in most MySQL installations. If you modify it directly, you can use it for some specialized purposes, such as to maintain a list of secure servers. For example, at TcX, the host table contains a list of all machines on the local network. These are granted all privileges.

You can also use the host table to indicate hosts that are *not* secure. Suppose that you have a machine `public.your.domain` that is located in a public area that you do not consider secure. You can allow access to all hosts on your network except that machine by using host table entries like this:

```
+-----+-----+-----+
| Host           | Db | ...
+-----+-----+-----+
| public.your.domain | % | ... (all privileges set to 'N')
| %.your.domain    | % | ... (all privileges set to 'Y')
+-----+-----+-----+
```

Naturally, you should always test your changes to the grant tables (for example, by using `SHOW GRANTS`) to make sure that your access privileges are actually set up the way you think they are.

5.8.7. When Privilege Changes Take Effect

When **mysqld** starts, it reads all grant table contents into memory. The in-memory tables become effective for access control at that point.

When the server reloads the grant tables, privileges for existing client connections are affected as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect at the next `USE db_name` statement.

Note: Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database or executing a `FLUSH PRIVILEGES` statement.

- Changes to global privileges and passwords take effect the next time the client connects.

If you modify the grant tables indirectly using statements such as GRANT, REVOKE, or SET PASSWORD, the server notices these changes and loads the grant tables into memory again immediately.

If you modify the grant tables directly using statements such as INSERT, UPDATE, or DELETE, your changes have no effect on privilege checking until you either restart the server or tell it to reload the tables. To reload the grant tables manually, issue a FLUSH PRIVILEGES statement or execute a **mysqladmin flush-privileges** or **mysqladmin reload** command.

If you change the grant tables directly but forget to reload them, your changes have *no effect* until you restart the server. This may leave you wondering why your changes do not seem to make any difference!

5.8.8. Causes of Access denied Errors

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not running, you cannot connect to it. For example, if you attempt to connect to the server and see a message such as one of those following, one cause might be that the server is not running:

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

It might also be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a --port option to indicate the proper port number, or a --socket option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```
shell> netstat -ln | grep mysql
```

- The grant tables must be properly set up so that the server can use them for

access control. For some distribution types (such as binary distributions on Windows, or RPM distributions on Linux), the installation process initializes the `mysql` database containing the grant tables. For distributions that do not do this, you must initialize the grant tables manually by running the `mysql_install_db` script. For details, see [Section 2.10.2, “Unix Post-Installation Procedures”](#).

One way to determine whether you need to initialize the grant tables is to look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If you do not, execute the `mysql_install_db` script. After running this script and starting the server, test the initial privileges by executing this command:

```
shell> mysql -u root test
```

The server should let you connect without error.

- After a fresh installation, you should connect to the server and set up your users and their access permissions:

```
shell> mysql -u root mysql
```

The server should let you connect because the MySQL `root` user has no password initially. That is also a security risk, so setting the password for the `root` accounts is something you should do while you're setting up your other MySQL accounts. For instructions on setting the initial passwords, see [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

- If you have updated an existing MySQL installation to a newer version, did you run the `mysql_upgrade` script? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).
- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

For information on how to deal with this, see [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#), and [Section A.2.3, “Client does not support authentication protocol”](#).

- If you try to connect as root and get the following error, it means that you do not have a row in the user table with a user column value of 'root' and that **mysqld** cannot resolve the hostname for your client:

```
Access denied for user ''@'unknown' to database mysql
```

In this case, you must restart the server with the `--skip-grant-tables` option and edit your `/etc/hosts` file or `\windows\hosts` file to add an entry for your host.

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you have not specified them on the command line, check your environment and any applicable option files. For example, if you get `Access denied` when you run a client without any options, make sure that you have not specified an old password in any of your option files!

You can suppress the use of option files by a client program by invoking it with the `--no-defaults` option. For example:

```
shell> mysqladmin --no-defaults -u root version
```

The option files that clients use are listed in [Section 4.3.2, “Using Option Files”](#). Environment variables are listed in [Appendix F, *Environment Variables*](#).

- If you get the following error, it means that you are using an incorrect root password:

```
shell> mysqladmin -u root -pXXXX ver
Access denied for user 'root'@'localhost' (using password: YES)
```

If the preceding error occurs even when you have not specified a password,

it means that you have an incorrect password listed in some option file. Try the `--no-defaults` option as described in the previous item.

For information on changing passwords, see [Section 5.9.5, “Assigning Account Passwords”](#).

If you have lost or forgotten the root password, you can restart **mysqld** with `--skip-grant-tables` to change the password. See [Section A.4.1, “How to Reset the Root Password”](#).

- If you change a password by using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must encrypt the password using the `PASSWORD()` function. If you do not use `PASSWORD()` for these statements, the password will not work. For example, the following statement sets a password, but fails to encrypt it, so the user is not able to connect afterward:

```
SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

Instead, set the password like this:

```
SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

The `PASSWORD()` function is unnecessary when you specify a password using the `GRANT` or (beginning with MySQL 5.0.2) `CREATE USER` statements, or the **mysqladmin password** command. Each of those automatically uses `PASSWORD()` to encrypt the password. See [Section 5.9.5, “Assigning Account Passwords”](#), and [Section 13.5.1.1, “CREATE USER Syntax”](#).

- `localhost` is a synonym for your local hostname, and is also the default host to which clients try to connect if you specify no host explicitly.

To avoid this problem on such systems, you can use a `--host=127.0.0.1` option to name the server host explicitly. This will make a TCP/IP connection to the local **mysqld** server. You can also use TCP/IP by specifying a `--host` option that uses the actual hostname of the local host. In this case, the hostname must be specified in a user table row on the server host, even though you are running the client program on the same host as the server.

- If you get an `Access denied` error when trying to connect to the database

with `mysql -u user_name`, you may have a problem with the user table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

```
SELECT * FROM user;
```

The result should include a row with the `Host` and `User` columns matching your computer's hostname and your MySQL username.

- The `Access denied` error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the user table that exactly matches the hostname and username that were given in the error message. For example, if you get an error message that contains `using password: NO`, it means that you tried to log in without a password.
- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the user table with a `Host` value that matches the client host:

```
Host ... is not allowed to connect to this MySQL server
```

You can fix this by setting up an account for the combination of client hostname and username that you are using when trying to connect.

If you do not know the IP number or hostname of the machine from which you are connecting, you should put a row with `'%'` as the `Host` column value in the user table. After trying to connect from the client machine, use a `SELECT USER()` query to see how you really did connect. (Then change the `'%'` in the user table row to the actual hostname that shows up in the log. Otherwise, your system is left insecure because it allows connections from any host for the given username.)

On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the `glibc` library than the one you are using. In this case, you should either upgrade your operating system or `glibc`, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

- If you specify a hostname when trying to connect, but get an error message where the hostname is not shown or is an IP number, it means that the MySQL server got an error when trying to resolve the IP number of the client host to a name:

```
shell> mysqladmin -u root -pxxxx -h some_hostname ver
Access denied for user 'root'@'' (using password: YES)
```

This indicates a DNS problem. To fix it, execute **mysqladmin flush-hosts** to reset the internal DNS hostname cache. See [Section 7.5.6, “How MySQL Uses DNS”](#).

Some permanent solutions are:

- Determine what is wrong with your DNS server and fix it.
 - Specify IP numbers rather than hostnames in the MySQL grant tables.
 - Put an entry for the client machine name in /etc/hosts or \windows\hosts.
 - Start **mysqld** with the `--skip-name-resolve` option.
 - Start **mysqld** with the `--skip-host-cache` option.
 - On Unix, if you are running the server and the client on the same machine, connect to localhost. Unix connections to localhost use a Unix socket file rather than TCP/IP.
 - On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the hostname . (period). Connections to . use a named pipe rather than TCP/IP.
- If `mysql -u root test` works but `mysql -h your_hostname -u root test` results in Access denied (where *your_hostname* is the actual hostname of the local host), you may not have the correct name for your host in the user table. A common problem here is that the Host value in the user table row specifies an unqualified hostname, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have an entry with host 'tcx' in the user table, but your DNS tells

MySQL that your hostname is 'tcx.subnet.se', the entry does not work. Try adding an entry to the user table that contains the IP number of your host as the Host column value. (Alternatively, you could add an entry to the user table with a Host value that contains a wildcard; for example, 'tcx.%'. However, use of hostnames ending with '%' is *insecure* and is *not* recommended!)

- If `mysql -u user_name test` works but `mysql -u user_name other_db_name` does not, you have not granted database access for `other_db_name` to the given user.
- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` does not work when executed on a remote client host, you have not enabled access to the server for the given username from the remote host.
- If you cannot figure out why you get Access denied, remove from the user table all entries that have Host values containing wildcards (entries that contain '%' or '_'). A very common error is to insert a new entry with `Host='%'` and `User='some_user'`, thinking that this allows you to specify localhost to connect from the same machine. The reason that this does not work is that the default privileges include an entry with `Host='localhost'` and `User=''`. Because that entry has a Host value 'localhost' that is more specific than '%', it is used in preference to the new entry when connecting from localhost! The correct procedure is to insert a second entry with `Host='localhost'` and `User='some_user'`, or to delete the entry with `Host='localhost'` and `User=''`. After deleting the entry, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables.
- If you get the following error, you may have a problem with the db or host table:

Access to database denied

If the entry selected from the db table has an empty value in the Host column, make sure that there are one or more corresponding entries in the host table specifying which hosts the db table entry applies to.

- If you are able to connect to the MySQL server, but get an Access denied message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA`

INFILE statement, your entry in the user table does not have the FILE privilege enabled.

- If you change the grant tables directly (for example, by using INSERT, UPDATE, or DELETE statements) and your changes seem to be ignored, remember that you must execute a FLUSH PRIVILEGES statement or a **mysqladmin flush-privileges** command to cause the server to re-read the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the root password with an UPDATE command, you won't need to specify the new password until after you flush the privileges, because the server won't know you've changed the password yet!
- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in [Section 5.8.7, “When Privilege Changes Take Effect”](#).
- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -pyour_pass db_name`. If you are able to connect using the **mysql** client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=your_pass` syntax to specify the password. If you use the `-p --password` option with no password value, MySQL prompts you for the password.)
- For testing, start the **mysqld** server with the `--skip-grant-tables` option. Then you can change the MySQL grant tables and use the **mysqlaccess** script to check whether your modifications have the desired effect. When you are satisfied with your changes, execute **mysqladmin flush-privileges** to tell the **mysqld** server to start using the new grant tables. (Reloading the grant tables overrides the `--skip-grant-tables` option. This enables you to tell the server to begin using the grant tables again without stopping and restarting it.)
- If everything else fails, start the **mysqld** server with a debugging option (for example, `--debug=d,general,query`). This prints host and user information about attempted connections, as well as information about each

command issued. See [Section E.1.2, “Creating Trace Files”](#).

- If you have any other problems with the MySQL grant tables and feel you must post the problem to the mailing list, always provide a dump of the MySQL grant tables. You can dump the tables with the **mysqldump mysql** command. To file a bug report, see the instructions at [Section 1.8, “How to Report Bugs or Problems”](#). In some cases, you may need to restart **mysqld** with `--skip-grant-tables` to run **mysqldump**.

5.8.9. Password Hashing as of MySQL 4.1

MySQL user accounts are listed in the `user` table of the `mysql` database. Each MySQL account is assigned a password, although what is stored in the `Password` column of the `user` table is not the plaintext version of the password, but a hash value computed from it. Password hash values are computed by the `PASSWORD()` function.

MySQL uses passwords in two phases of client/server communication:

- When a client attempts to connect to the server, there is an initial authentication step in which the client must present a password that has a hash value matching the hash value stored in the `user` table for the account that the client wants to use.
- After the client connects, it can (if it has sufficient privileges) set or change the password hashes for accounts listed in the `user` table. The client can do this by using the `PASSWORD()` function to generate a password hash, or by using the `GRANT` or `SET PASSWORD` statements.

In other words, the server *uses* hash values during authentication when a client first attempts to connect. The server *generates* hash values if a connected client invokes the `PASSWORD()` function or uses a `GRANT` or `SET PASSWORD` statement to set or change a password.

The password hashing mechanism was updated in MySQL 4.1 to provide better security and to reduce the risk of passwords being intercepted. However, this new mechanism is understood only by MySQL 4.1 (and newer) servers and clients, which can result in some compatibility problems. A 4.1 or newer client can connect to a pre-4.1 server, because the client understands both the old and

new password hashing mechanisms. However, a pre-4.1 client that attempts to connect to a 4.1 or newer server may run into difficulties. For example, a 3.23 **mysql** client that attempts to connect to a 5.0 server may fail with the following error message:

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

Another common example of this phenomenon occurs for attempts to use the older PHP `mysql` extension after upgrading to MySQL 4.1 or newer. (See [Section 22.3.1, “Common Problems with MySQL and PHP”](#).)

The following discussion describes the differences between the old and new password mechanisms, and what you should do if you upgrade your server but need to maintain backward compatibility with pre-4.1 clients. Additional information can be found in [Section A.2.3, “Client does not support authentication protocol”](#). This information is of particular importance to PHP programmers migrating MySQL databases from version 4.0 or lower to version 4.1 or higher.

Note: This discussion contrasts 4.1 behavior with pre-4.1 behavior, but the 4.1 behavior described here actually begins with 4.1.1. MySQL 4.1.0 is an “odd” release because it has a slightly different mechanism than that implemented in 4.1.1 and up. Differences between 4.1.0 and more recent versions are described further in MySQL 3.23, 4.0, 4.1 Reference Manual.

Prior to MySQL 4.1, password hashes computed by the `PASSWORD()` function are 16 bytes long. Such hashes look like this:

```
mysql> SELECT PASSWORD( 'mypass' );
+-----+
| PASSWORD( 'mypass' ) |
+-----+
| 6f8c114b58f2ce9e    |
+-----+
```

The Password column of the user table (in which these hashes are stored) also is 16 bytes long before MySQL 4.1.

As of MySQL 4.1, the `PASSWORD()` function has been modified to produce a longer 41-byte hash value:

```
mysql> SELECT PASSWORD( 'mypass' );
+-----+
| PASSWORD( 'mypass' ) |
+-----+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-----+
```

Accordingly, the Password column in the user table also must be 41 bytes long to store these values:

- If you perform a new installation of MySQL 5.0, the Password column is made 41 bytes long automatically.
- Upgrading from MySQL 4.1 (4.1.1 or later in the 4.1 series) to MySQL 5.0 should not give rise to any issues in this regard because both versions use the same password hashing mechanism. If you wish to upgrade an older release of MySQL to version 5.0, you should upgrade to version 4.1 first, then upgrade the 4.1 installation to 5.0.

A widened Password column can store password hashes in both the old and new formats. The format of any given password hash value can be determined two ways:

- The obvious difference is the length (16 bytes versus 41 bytes).
- A second difference is that password hashes in the new format always begin with a '*' character, whereas passwords in the old format never do.

The longer password hash format has better cryptographic properties, and client authentication based on long hashes is more secure than that based on the older short hashes.

The differences between short and long password hashes are relevant both for how the server uses passwords during authentication and for how it generates password hashes for connected clients that perform password-changing operations.

The way in which the server uses password hashes during authentication is affected by the width of the Password column:

- If the column is short, only short-hash authentication is used.

- If the column is long, it can hold either short or long hashes, and the server can use either format:
 - Pre-4.1 clients can connect, although because they know only about the old hashing mechanism, they can authenticate only using accounts that have short hashes.
 - 4.1 and later clients can authenticate using accounts that have short or long hashes.

Even for short-hash accounts, the authentication process is actually a bit more secure for 4.1 and later clients than for older clients. In terms of security, the gradient from least to most secure is:

- Pre-4.1 client authenticating with short password hash
- 4.1 or later client authenticating with short password hash
- 4.1 or later client authenticating with long password hash

The way in which the server generates password hashes for connected clients is affected by the width of the Password column and by the `--old-passwords` option. A 4.1 or later server generates long hashes only if certain conditions are met: The Password column must be wide enough to hold long values and the `--old-passwords` option must not be given. These conditions apply as follows:

- The Password column must be wide enough to hold long hashes (41 bytes). If the column has not been updated and still has the pre-4.1 width of 16 bytes, the server notices that long hashes cannot fit into it and generates only short hashes when a client performs password-changing operations using `PASSWORD()`, `GRANT`, or `SET PASSWORD`. This is the behavior that occurs if you have upgraded to 4.1 but have not yet run the **`mysql_fix_privilege_tables`** script to widen the Password column.
- If the Password column is wide, it can store either short or long password hashes. In this case, `PASSWORD()`, `GRANT`, and `SET PASSWORD` generate long hashes unless the server was started with the `--old-passwords` option. That option forces the server to generate short password hashes instead.

The purpose of the `--old-passwords` option is to enable you to maintain

backward compatibility with pre-4.1 clients under circumstances where the server would otherwise generate long password hashes. The option doesn't affect authentication (4.1 and later clients can still use accounts that have long password hashes), but it does prevent creation of a long password hash in the user table as the result of a password-changing operation. Were that to occur, the account no longer could be used by pre-4.1 clients. Without the `--old-passwords` option, the following undesirable scenario is possible:

- An old client connects to an account that has a short password hash.
- The client changes its own password. Without `--old-passwords`, this results in the account having a long password hash.
- The next time the old client attempts to connect to the account, it cannot, because the account has a long password hash that requires the new hashing mechanism during authentication. (Once an account has a long password hash in the user table, only 4.1 and later clients can authenticate for it, because pre-4.1 clients do not understand long hashes.)

This scenario illustrates that, if you must support older pre-4.1 clients, it is dangerous to run a 4.1 or newer server without using the `--old-passwords` option. By running the server with `--old-passwords`, password-changing operations do not generate long password hashes and thus do not cause accounts to become inaccessible to older clients. (Those clients cannot inadvertently lock themselves out by changing their password and ending up with a long password hash.)

The downside of the `--old-passwords` option is that any passwords you create or change use short hashes, even for 4.1 clients. Thus, you lose the additional security provided by long password hashes. If you want to create an account that has a long hash (for example, for use by 4.1 clients), you must do so while running the server without `--old-passwords`.

The following scenarios are possible for running a 4.1 or later server:

Scenario 1: Short Password column in user table:

- Only short hashes can be stored in the Password column.
- The server uses only short hashes during client authentication.

- For connected clients, password hash-generating operations involving `PASSWORD()`, `GRANT`, or `SET PASSWORD` use short hashes exclusively. Any change to an account's password results in that account having a short password hash.
- The `--old-passwords` option can be used but is superfluous because with a short Password column, the server generates only short password hashes anyway.

Scenario 2: Long Password column; server not started with `--old-passwords` option:

- Short or long hashes can be stored in the Password column.
- 4.1 and later clients can authenticate using accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only using accounts that have short hashes.
- For connected clients, password hash-generating operations involving `PASSWORD()`, `GRANT`, or `SET PASSWORD` use long hashes exclusively. A change to an account's password results in that account having a long password hash.

As indicated earlier, a danger in this scenario is that it is possible for accounts that have a short password hash to become inaccessible to pre-4.1 clients. A change to such an account's password made via `GRANT`, `PASSWORD()`, or `SET PASSWORD` results in the account being given a long password hash. From that point on, no pre-4.1 client can authenticate to that account until the client upgrades to 4.1.

To deal with this problem, you can change a password in a special way. For example, normally you use `SET PASSWORD` as follows to change an account password:

```
SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

To change the password but create a short hash, use the `OLD_PASSWORD()` function instead:

```
SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

OLD_PASSWORD() is useful for situations in which you explicitly want to generate a short hash.

Scenario 3: Long Password column; 4.1 or newer server started with --old-passwords option:

- Short or long hashes can be stored in the Password column.
- 4.1 and later clients can authenticate for accounts that have short or long hashes (but note that it is possible to create long hashes only when the server is started without --old-passwords).
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving PASSWORD(), GRANT, or SET PASSWORD use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

In this scenario, you cannot create accounts that have long password hashes, because the --old-passwords option prevents generation of long hashes. Also, if you create an account with a long hash before using the --old-passwords option, changing the account's password while --old-passwords is in effect results in the account being given a short password, causing it to lose the security benefits of a longer hash.

The disadvantages for these scenarios may be summarized as follows:

In scenario 1, you cannot take advantage of longer hashes that provide more secure authentication.

In scenario 2, accounts with short hashes become inaccessible to pre-4.1 clients if you change their passwords without explicitly using OLD_PASSWORD().

In scenario 3, --old-passwords prevents accounts with short hashes from becoming inaccessible, but password-changing operations cause accounts with long hashes to revert to short hashes, and you cannot change them back to long hashes while --old-passwords is in effect.

5.8.9.1. Implications of Password Hashing Changes for Application Programs

An upgrade to MySQL version 4.1 or later can cause compatibility issues for applications that use `PASSWORD()` to generate passwords for their own purposes. Applications really should not do this, because `PASSWORD()` should be used only to manage passwords for MySQL accounts. But some applications use `PASSWORD()` for their own purposes anyway.

If you upgrade to 4.1 or later from a pre-4.1 version of MySQL and run the server under conditions where it generates long password hashes, an application using `PASSWORD()` for its own passwords breaks. The recommended course of action in such cases is to modify the application to use another function, such as `SHA1()` or `MD5()`, to produce hashed values. If that is not possible, you can use the `OLD_PASSWORD()` function, which is provided for generate short hashes in the old format. However, you should note that `OLD_PASSWORD()` may one day no longer be supported.

If the server is running under circumstances where it generates short hashes, `OLD_PASSWORD()` is available but is equivalent to `PASSWORD()`.

PHP programmers migrating their MySQL databases from version 4.0 or lower to version 4.1 or higher should see [Section 22.3, “MySQL PHP API”](#).

5.9. MySQL User Account Management

This section describes how to set up accounts for clients of your MySQL server. It discusses the following topics:

- The meaning of account names and passwords as used in MySQL and how that compares to names and passwords used by your operating system
- How to set up new accounts and remove existing accounts
- How to change passwords
- Guidelines for using passwords securely
- How to use secure connections with SSL

5.9.1. MySQL Usernames and Passwords

A MySQL account is defined in terms of a username and the client host or hosts from which the user can connect to the server. The account also has a password. There are several distinctions between the way usernames and passwords are used by MySQL and the way they are used by your operating system:

- Usernames, as used by MySQL for authentication purposes, have nothing to do with usernames (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix username as the MySQL username, but that is for convenience only. The default can be overridden easily, because client programs allow any username to be specified with a `-u` or `--user` option. Because this means that anyone can attempt to connect to the server using any username, you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a username for an account that has no password is able to connect successfully to the server.
- MySQL usernames can be up to a maximum of 16 characters long. This limit is hard-coded in the MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database *does not work*.

Note: You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure prescribed by MySQL AB that is described in [Section 5.6.2, “mysql upgrade — Check Tables for MySQL Upgrade”](#). Attempting to redefine MySQL's system tables in any other fashion results in undefined (and unsupported!) behavior.

Operating system usernames are completely unrelated to MySQL usernames and may even be of a different maximum length. For example, Unix usernames typically are limited to eight characters.

- MySQL passwords have nothing to do with passwords for logging in to your operating system. There is no necessary connection between the password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.
- MySQL encrypts passwords using its own algorithm. This encryption is different from that used during the Unix login process. MySQL password encryption is the same as that implemented by the `PASSWORD()` SQL function. Unix password encryption is the same as that implemented by the `ENCRYPT()` SQL function. See the descriptions of the `PASSWORD()` and `ENCRYPT()` functions in [Section 12.9.2, “Encryption and Compression Functions”](#). From version 4.1 on, MySQL employs a stronger authentication method that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. (In earlier versions, even though passwords are stored in encrypted form in the user table, knowledge of the encrypted password value could be used to connect to the MySQL server.)

When you install MySQL, the grant tables are populated with an initial set of accounts. These accounts have names and access privileges that are described in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#), which also discusses how to assign passwords to them. Thereafter, you normally set up, modify, and remove MySQL accounts using statements such as `GRANT` and `REVOKE`. See [Section 13.5.1, “Account Management Statements”](#).

When you connect to a MySQL server with a command-line client, you should specify the username and password for the account that you want to use:

```
shell> mysql --user=monty --password=guess db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u monty -pguess db_name
```

There must be *no space* between the `-p` option and the following password value. See [Section 5.8.4, “Connecting to the MySQL Server”](#).

The preceding commands include the password value on the command line, which can be a security risk. See [Section 5.9.6, “Keeping Your Password Secure”](#). To avoid this problem, specify the `--password` or `-p` option without any following password value:

```
shell> mysql --user=monty --password db_name
shell> mysql -u monty -p db_name
```

When the password option has no password value, the client program prints a prompt and waits for you to enter the password. (In these examples, `db_name` is *not* interpreted as a password because it is separated from the preceding password option by a space.)

On some systems, the library routine that MySQL uses to prompt for a password automatically limits the password to eight characters. That is a problem with the system library, not with MySQL. Internally, MySQL doesn't have any limit for the length of the password. To work around the problem, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

5.9.2. Adding New User Accounts to MySQL

You can create MySQL accounts in two ways:

- By using statements intended for creating accounts, such as `CREATE USER` or `GRANT`
- By manipulating the MySQL grant tables directly with statements such as `INSERT`, `UPDATE`, or `DELETE`

The preferred method is to use account-creation statements because they are more concise and less error-prone. `CREATE USER` and `GRANT` are described in [Section 13.5.1.1, “CREATE USER Syntax”](#), and [Section 13.5.1.3, “GRANT Syntax”](#).

Another option for creating accounts is to use one of several available third-party programs that offer capabilities for MySQL account administration. phpMyAdmin is one such program.

The following examples show how to use the **mysql** client program to set up new users. These examples assume that privileges are set up according to the defaults described in [Section 2.10.3, “Securing the Initial MySQL Accounts”](#). This means that to make changes, you must connect to the MySQL server as the MySQL root user, and the root account must have the INSERT privilege for the mysql database and the RELOAD administrative privilege.

First, use the **mysql** program to connect to the server as the MySQL root user:

```
shell> mysql --user=root mysql
```

If you have assigned a password to the root account, you'll also need to supply a `--password` or `-p` option for this **mysql** command and also for those later in this section.

After connecting to the server as root, you can add new accounts. The following statements use GRANT to set up four new accounts:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'
-> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'
-> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
mysql> GRANT USAGE ON *.* TO 'dummy'@'localhost';
```

The accounts created by these GRANT statements have the following properties:

- Two of the accounts have a username of monty and a password of some_pass. Both accounts are superuser accounts with full privileges to do anything. One account ('monty'@'localhost') can be used only when connecting from the local host. The other ('monty'@'%') can be used to connect from any other host. Note that it is necessary to have both accounts for monty to be able to connect from anywhere as monty. Without the localhost account, the anonymous-user account for localhost that is created by **mysql_install_db** would take precedence when monty connects from the local host. As a result, monty would be treated as an anonymous user. The reason for this is that the anonymous-user account has a more

specific Host column value than the 'monty'@'%' account and thus comes earlier in the user table sort order. (user table sorting is discussed in [Section 5.8.5, “Access Control, Stage 1: Connection Verification”](#).)

- One account has a username of admin and no password. This account can be used only by connecting from the local host. It is granted the RELOAD and PROCESS administrative privileges. These privileges allow the admin user to execute the **mysqladmin reload**, **mysqladmin refresh**, and **mysqladmin flush-xxx** commands, as well as **mysqladmin processlist** . No privileges are granted for accessing any databases. You could add such privileges later by issuing additional GRANT statements.
- One account has a username of dummy and no password. This account can be used only by connecting from the local host. No privileges are granted. The USAGE privilege in the GRANT statement enables you to create an account without giving it any privileges. It has the effect of setting all the global privileges to 'N'. It is assumed that you will grant specific privileges to the account later.

As an alternative to GRANT, you can create the same accounts directly by issuing INSERT statements and then telling the server to reload the grant tables using FLUSH PRIVILEGES:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user
->     VALUES('localhost','monty',PASSWORD('some_pass'),
->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
->     VALUES('%','monty',PASSWORD('some_pass'),
->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user SET Host='localhost',User='admin',
->     Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User>Password)
->     VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

The reason for using FLUSH PRIVILEGES when you create accounts with INSERT is to tell the server to re-read the grant tables. Otherwise, the changes go unnoticed until you restart the server. With GRANT, FLUSH PRIVILEGES is unnecessary.

The reason for using the PASSWORD() function with INSERT is to encrypt the

password. The GRANT statement encrypts the password for you, so PASSWORD() is unnecessary.

The 'Y' values enable privileges for the accounts. Depending on your MySQL version, you may have to use a different number of 'Y' values in the first two INSERT statements. For the admin account, you may also employ the more readable extended INSERT syntax using SET.

In the INSERT statement for the dummy account, only the Host, User, and Password columns in the user table row are assigned values. None of the privilege columns are set explicitly, so MySQL assigns them all the default value of 'N'. This is equivalent to what GRANT USAGE does.

Note that to set up a superuser account, it is necessary only to create a user table entry with the privilege columns set to 'Y'. user table privileges are global, so no entries in any of the other grant tables are needed.

The next examples create three accounts and give them access to specific databases. Each of them has a username of custom and password of obscure.

To create the accounts with GRANT, use the following statements:

```
shell> mysql --user=root mysql
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP
-> ON bankaccount.*
-> TO 'custom'@'localhost'
-> IDENTIFIED BY 'obscure';
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP
-> ON expenses.*
-> TO 'custom'@'whitehouse.gov'
-> IDENTIFIED BY 'obscure';
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP
-> ON customer.*
-> TO 'custom'@'server.domain'
-> IDENTIFIED BY 'obscure';
```

The three accounts can be used as follows:

- The first account can access the bankaccount database, but only from the local host.
- The second account can access the expenses database, but only from the host whitehouse.gov.

- The third account can access the customer database, but only from the host `server.domain`.

To set up the custom accounts without GRANT, use INSERT statements as follows to modify the grant tables directly:

```

shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
->     VALUES('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
->     VALUES('whitehouse.gov','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
->     VALUES('server.domain','custom',PASSWORD('obscure'));
mysql> INSERT INTO db
->     (Host,Db,User,Select_priv,Insert_priv,
->     Update_priv,Delete_priv,Create_priv,Drop_priv)
->     VALUES('localhost','bankaccount','custom',
->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
->     (Host,Db,User,Select_priv,Insert_priv,
->     Update_priv,Delete_priv,Create_priv,Drop_priv)
->     VALUES('whitehouse.gov','expenses','custom',
->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
->     (Host,Db,User,Select_priv,Insert_priv,
->     Update_priv,Delete_priv,Create_priv,Drop_priv)
->     VALUES('server.domain','customer','custom',
->     'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;

```

The first three INSERT statements add user table entries that allow the user `custom` to connect from the various hosts with the given password, but grant no global privileges (all privileges are set to the default value of 'N'). The next three INSERT statements add db table entries that grant privileges to `custom` for the `bankaccount`, `expenses`, and `customer` databases, but only when accessed from the proper hosts. As usual when you modify the grant tables directly, you must tell the server to reload them with `FLUSH PRIVILEGES` so that the privilege changes take effect.

If you want to give a specific user access from all machines in a given domain (for example, `mydomain.com`), you can issue a GRANT statement that uses the '%' wildcard character in the host part of the account name:

```

mysql> GRANT ...
->     ON *.*

```

```
->     TO 'myname'@'%.mydomain.com'  
->     IDENTIFIED BY 'mypass';
```

To do the same thing by modifying the grant tables directly, do this:

```
mysql> INSERT INTO user (Host,User,Password,...)  
->     VALUES('%.mydomain.com','myname',PASSWORD('mypass'),...);  
mysql> FLUSH PRIVILEGES;
```

5.9.3. Removing User Accounts from MySQL

To remove an account, use the `DROP USER` statement, which is described in [Section 13.5.1.2, “DROP USER Syntax”](#).

5.9.4. Limiting Account Resources

One means of limiting use of MySQL server resources is to set the `max_user_connections` system variable to a non-zero value. However, this method is strictly global, and does not allow for management of individual accounts. In addition, it limits only the number of simultaneous connections made using a single account, and not what a client can do once connected. Both types of control are interest to many MySQL administrators, particularly those working for Internet Service Providers.

In MySQL 5.0, you can limit the following server resources for individual accounts:

- The number of queries that an account can issue per hour
- The number of updates that an account can issue per hour
- The number of times an account can connect to the server per hour

Any statement that a client can issue counts against the query limit. Only statements that modify databases or tables count against the update limit.

From MySQL 5.0.3 on, it is also possible to limit the number of simultaneous connections to the server on a per-account basis.

An account in this context is a single row in the `user` table. Each account is uniquely identified by its `user` and `Host` column values.

As a prerequisite for using this feature, the user table in the mysql database must contain the resource-related columns. Resource limits are stored in the max_questions, max_updates, max_connections, and max_user_connections columns. If your user table doesn't have these columns, it must be upgraded; see [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

To set resource limits with a GRANT statement, use a WITH clause that names each resource to be limited and a per-hour count indicating the limit value. For example, to create a new account that can access the customer database, but only in a limited fashion, issue this statement:

```
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'  
-> IDENTIFIED BY 'frank'  
-> WITH MAX_QUERIES_PER_HOUR 20  
-> MAX_UPDATES_PER_HOUR 10  
-> MAX_CONNECTIONS_PER_HOUR 5  
-> MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the WITH clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. If the GRANT statement has no WITH clause, the limits are each set to the default value of zero (that is, no limit). For MAX_USER_CONNECTIONS, the limit is an integer indicating the maximum number of simultaneous connections the account can make at any one time. If the limit is set to the default value of zero, the max_user_connections system variable determines the number of simultaneous connections for the account.

To set or change limits for an existing account, use a GRANT USAGE statement at the global level (ON *.*). The following statement changes the query limit for francis to 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'  
-> WITH MAX_QUERIES_PER_HOUR 100;
```

This statement leaves the account's existing privileges unchanged and modifies only the limit values specified.

To remove an existing limit, set its value to zero. For example, to remove the limit on how many times per hour francis can connect, use this statement:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'  
-> WITH MAX_CONNECTIONS_PER_HOUR 0;
```

Resource-use counting takes place when any account has a non-zero limit placed on its use of any of the resources.

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, further connections for the account are rejected until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, further queries or updates are rejected until the hour is up. In all such cases, an appropriate error message is issued.

Resource counting is done per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a **mysqladmin reload** command).
- The counts for an individual account can be set to zero by re-granting it any of its limits. To do this, use `GRANT USAGE` as described earlier and specify a limit value equal to the value that the account currently has.

Counter resets do not affect the `MAX_USER_CONNECTIONS` limit.

All counts begin at zero when the server starts; counts are not carried over through a restart.

5.9.5. Assigning Account Passwords

Passwords may be assigned from the command line by using the **mysqladmin** command:

```
shell> mysqladmin -u user_name -h host_name password "newpwd"
```

The account for which this command resets the password is the one with a user

table row that matches *user_name* in the User column and the client host *from which you connect* in the Host column.

Another way to assign a password to an account is to issue a SET PASSWORD statement:

```
mysql> SET PASSWORD FOR 'jeffrey'@'%' = PASSWORD('biscuit');
```

Only users such as root that have update access to the mysql database can change the password for other users. If you are not connected as an anonymous user, you can change your own password by omitting the FOR clause:

```
mysql> SET PASSWORD = PASSWORD('biscuit');
```

You can also use a GRANT USAGE statement at the global level (ON *.*) to assign a password to an account without affecting the account's current privileges:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
```

Although it is generally preferable to assign passwords using one of the preceding methods, you can also do so by modifying the user table directly:

- To establish a password when creating a new account, provide a value for the Password column:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
  -> VALUES('%','jeffrey',PASSWORD('biscuit'));
mysql> FLUSH PRIVILEGES;
```

- To change the password for an existing account, use UPDATE to set the Password column value:

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('bagel')
  -> WHERE Host = '%' AND User = 'francis';
mysql> FLUSH PRIVILEGES;
```

When you assign an account a non-empty password using SET PASSWORD, INSERT, or UPDATE, you must use the PASSWORD() function to encrypt it. PASSWORD() is necessary because the user table stores passwords in encrypted form, not as plaintext. If you forget that fact, you are likely to set passwords like this:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
-> VALUES('%','jeffrey','biscuit');
mysql> FLUSH PRIVILEGES;
```

The result is that the literal value 'biscuit' is stored as the password in the user table, not the encrypted value. When jeffrey attempts to connect to the server using this password, the value is encrypted and compared to the value stored in the user table. However, the stored value is the literal string 'biscuit', so the comparison fails and the server rejects the connection:

```
shell> mysql -u jeffrey -pbiscuit test
Access denied
```

If you assign passwords using the GRANT ... IDENTIFIED BY statement or the **mysqladmin password** command, they both take care of encrypting the password for you. In these cases, using PASSWORD() function is unnecessary.

Note: PASSWORD() encryption is different from Unix password encryption. See [Section 5.9.1, “MySQL Usernames and Passwords”](#).

5.9.6. Keeping Your Password Secure

On an administrative level, you should never grant access to the user grant table to any non-administrative accounts.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method:

- Use a -pyour_pass or --password=your_pass option on the command line. For example:

```
shell> mysql -u francis -pfrank db_name
```

This is convenient *but insecure*, because your password becomes visible to system status programs such as **ps** that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. On

some systems this strategy is ineffective, anyway, and the password remains visible to **ps**. (SystemV Unix systems and perhaps others are subject to this problem.)

- Use the `-p` or `--password` option with no password value specified. In this case, the client program solicits the password from the terminal:

```
shell> mysql -u francis -p db_name
Enter password: *****
```

The ‘*’ characters indicate where you enter your password. The password is not displayed as you enter it.

It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs non-interactively, there is no opportunity to enter the password from the terminal. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- Store your password in an option file. For example, on Unix you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

```
[client]
password=your_pass
```

If you store your password in `.my.cnf`, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

```
shell> chmod 600 .my.cnf
```

[Section 4.3.2, “Using Option Files”](#), discusses option files in more detail.

- Store your password in the `MYSQL_PWD` environment variable. This method of specifying your MySQL password must be considered *extremely insecure* and should not be used. Some versions of **ps** include an option to display the environment of running processes. If you set `MYSQL_PWD`, your password is exposed to any other user who runs **ps**. Even on systems

without such a version of **ps**, it is unwise to assume that there are no other methods by which users can examine process environments. See [Appendix F, *Environment Variables*](#).

All in all, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

5.9.7. Using Secure Connections

MySQL supports secure (encrypted) connections between MySQL clients and the server using the Secure Sockets Layer (SSL) protocol. This section discusses how to use SSL connections. It also describes a way to set up SSH on Windows. For information on requiring users to use SSL connections, see [Section 13.5.1.3, “GRANT Syntax”](#).

The standard configuration of MySQL is intended to be as fast as possible, so encrypted connections are not used by default. Doing so would make the client/server protocol much slower. Encrypting data is a CPU-intensive operation that requires the computer to do additional work and can delay other MySQL tasks. For applications that require the security provided by encrypted connections, the extra computation is warranted.

MySQL allows encryption to be enabled on a per-connection basis. You can choose a normal unencrypted connection or a secure encrypted SSL connection according to the requirements of individual applications.

Secure connections are based on the OpenSSL API and are available through the MySQL C API. Replication uses the C API, so secure connections can be used between master and slave servers.

5.9.7.1. Basic SSL Concepts

To understand how MySQL uses SSL, it is necessary to explain some basic SSL and X509 concepts. People who are familiar with these can skip this part of the discussion.

By default, MySQL uses unencrypted connections between the client and the server. This means that someone with access to the network could watch all your traffic and look at the data being sent or received. They could even change the

data while it is in transit between client and server. To improve security a little, you can compress client/server traffic by using the `--compress` option when invoking client programs. However, this does not foil a determined attacker.

When you need to move information over a network in a secure fashion, an unencrypted connection is unacceptable. Encryption is the way to make any kind of data unreadable. In fact, today's practice requires many additional security elements from encryption algorithms. They should resist many kind of known attacks such as changing the order of encrypted messages or replaying data twice.

SSL is a protocol that uses different encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect any data change, loss, or replay. SSL also incorporates algorithms that provide identity verification using the X509 standard.

X509 makes it possible to identify someone on the Internet. It is most commonly used in e-commerce applications. In basic terms, there should be some company called a "Certificate Authority" (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can show the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted with this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

If you need more information about SSL, X509, or encryption, use your favorite Internet search engine to search for the keywords in which you are interested.

5.9.7.2. Using SSL Connections

To use SSL connections between the MySQL server and client programs, your system must support either OpenSSL or yaSSL and your version of MySQL must be built with SSL support.

To make it easier to use secure connections, MySQL is bundled with yaSSL as of MySQL 5.0.10. (MySQL and yaSSL employ the same licensing model, whereas OpenSSL uses an Apache-style license.) yaSSL support initially was available only for a few platforms, but now it is available on all platforms supported by

MySQL AB.

To get secure connections to work with MySQL and SSL, you must do the following:

1. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, and you are going to use OpenSSL rather than the bundled yaSSL library, install OpenSSL if it has not already been installed. We have tested MySQL with OpenSSL 0.9.6. To obtain OpenSSL, visit <http://www.openssl.org>.
2. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, configure a MySQL source distribution to use SSL. When you configure MySQL, invoke the **configure** script with the appropriate option to select the SSL library that you want to use.

For yaSSL:

```
shell> ./configure --with-yassl
```

For OpenSSL:

```
shell> ./configure --with-openssl
```

Before MySQL 5.0, it was also necessary to use `--with-vio`, but that option is no longer required.

Note that yaSSL support on Unix platforms requires that either `/dev/urandom` or `/dev/random` be installed to retrieve true random numbers. For additional information (especially regarding yaSSL on Solaris versions prior to 2.8 and HP-UX), see Bug #13164.

3. Make sure that you have upgraded your grant tables to include the SSL-related columns in the `mysql.user` table. This is necessary if your grant tables date from a version of MySQL older than 4.0. The upgrade procedure is described in [Section 5.6.2, “mysql upgrade — Check Tables for MySQL Upgrade”](#).
4. To check whether a server binary is compiled with SSL support, invoke it with the `--ssl` option. An error will occur if the server does not support SSL:

```
shell> mysqld --ssl --help
060525 14:18:52 [ERROR] mysqld: unknown option '--ssl'
```

To check whether a running **mysqld** server supports SSL, examine the value of the `have_openssl` system variable:

```
mysql> SHOW VARIABLES LIKE 'have_openssl';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_openssl  | YES   |
+-----+-----+
```

If the value is YES, the server supports SSL connections. If the value is DISABLED, the server supports SSL connections but was not started with the appropriate `--ssl-xxx` options (described later in this section). If the value is YES, the server supports SSL connections.

To start the MySQL server so that it allows clients to connect via SSL, use the options that identify the key and certificate files the server needs when establishing a secure connection:

```
shell> mysqld --ssl-ca=cacert.pem \
          --ssl-cert=server-cert.pem \
          --ssl-key=server-key.pem
```

- `--ssl-ca` identifies the Certificate Authority (CA) certificate.
- `--ssl-cert` identifies the server public key. This can be sent to the client and authenticated against the CA certificate that it has.
- `--ssl-key` identifies the server private key.

To establish a secure connection to a MySQL server with yaSSL support, start a client like this:

```
shell> mysql --ssl-ca=cacert.pem \
           --ssl-cert=client-cert.pem \
           --ssl-key=client-key.pem
```

In other words, the options are similar to those used for the server. Note that the Certificate Authority certificate has to be the same.

A client can determine whether the current connection with the server uses SSL by checking the value of the `Ssl_cipher` status variable. The value of `Ssl_cipher` is non-empty if SSL is used, and empty otherwise. For example:

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+-----+-----+
| Variable_name | Value                |
+-----+-----+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+-----+-----+
```

For the **mysql** client, you can use the `STATUS` or `\s` command and check the SSL line:

```
mysql> \s
...
SSL:                Not in use
...
```

Or:

```
mysql> \s
...
SSL:                Cipher in use is DHE-RSA-AES256-SHA
...
```

To establish a secure connection from within an application program, use the `mysql_ssl_set()` C API function to set the appropriate certificate options before calling `mysql_real_connect()`. See [Section 22.2.3.66, “mysql_ssl_set\(\)”](#).

5.9.7.3. SSL Command Options

The following list describes options that are used for specifying the use of SSL, certificate files, and key files. They can be given on the command line or in an option file.

These options are not available unless MySQL has been built with SSL support. See [Section 5.9.7.2, “Using SSL Connections”](#).

- `--ssl`

For the server, this option specifies that the server allows SSL connections. For a client program, it allows the client to connect to the server using SSL.

This option is not sufficient in itself to cause an SSL connection to be used. You must also specify the `--ssl-ca`, `--ssl-cert`, and `--ssl-key` options.

This option is more often used in its opposite form to override any other SSL options and indicate that SSL should *not* be used. To do this, specify the option as `--skip-ssl` or `--ssl=0`.

Note that use of `--ssl` does not *require* an SSL connection. For example, if the server or client is compiled without SSL support, a normal unencrypted connection is used.

The secure way to ensure that an SSL connection is used is to create an account on the server that includes a `REQUIRE SSL` clause in the `GRANT` statement. Then use this account to connect to the server, with both a server and client that have SSL support enabled.

- `--ssl-ca=file_name`

The path to a file with a list of trusted SSL CAs.

- `--ssl-capath=directory_name`

The path to a directory that contains trusted SSL CA certificates in PEM format.

- `--ssl-cert=file_name`

The name of the SSL certificate file to use for establishing a secure connection.

- `--ssl-cipher=cipher_list`

A list of allowable ciphers to use for SSL encryption. *cipher_list* has the same format as the `openssl ciphers` command.

Example: `--ssl-cipher=ALL:-AES:-EXP`

- `--ssl-key=file_name`

The name of the SSL key file to use for establishing a secure connection.

- `--ssl-verify-server-cert`

This option is available for client programs. It causes the server's Common Name value in its certificate to be verified against the hostname used when connecting to the server, and the connection is rejected if there is a mismatch. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default. This option was added in MySQL 5.0.23.

5.9.7.4. Setting Up SSL Certificates for MySQL

Here is an example of setting up SSL certificates for MySQL using OpenSSL:

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/cacert.pem
        -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
```

```
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL A
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..+++++
# .....+++++
# writing new private key to '/home/monty/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL A
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
```

```
# Remove the passphrase from the key (optional)
#

openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca -policy policy_anything -out $DIR/server-cert.pem \
    -config $DIR/openssl.cnf -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName          :PRINTABLE:'FI'
# organizationName     :PRINTABLE:'MySQL AB'
# commonName           :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
    $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
```

```
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL A
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove a passphrase from the key (optional)
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca -policy policy_anything -out $DIR/client-cert.pem \
    -config $DIR/openssl.cnf -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName          :PRINTABLE:'FI'
# organizationName     :PRINTABLE:'MySQL AB'
# commonName           :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#
```

```

cnf=""
cnf="$cnf [client]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/client-cert.pem"
cnf="$cnf ssl-key=$DIR/client-key.pem"
cnf="$cnf [mysqld]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/server-cert.pem"
cnf="$cnf ssl-key=$DIR/server-key.pem"
echo $cnf | replace " " '
' > $DIR/my.cnf

```

To test SSL connections, start the server as follows, where \$DIR is the pathname to the directory where the sample my.cnf option file is located:

```
shell> mysqld --defaults-file=$DIR/my.cnf &
```

Then invoke a client program using the same option file:

```
shell> mysql --defaults-file=$DIR/my.cnf
```

If you have a MySQL source distribution, you can also test your setup by modifying the preceding my.cnf file to refer to the demonstration certificate and key files in the SSL directory of the distribution.

5.9.7.5. Connecting to MySQL Remotely from Windows with SSH

Here is a note that describes how to get a secure connection to a remote MySQL server with SSH (by David Carlson <dcarlson@mplcomm.com>):

1. Install an SSH client on your Windows machine. As a user, the best non-free one I have found is from SecureCRT from <http://www.vandyke.com/>. Another option is f-secure from <http://www.f-secure.com/>. You can also find some free ones on Google at http://directory.google.com/Top/Computers/Security/Products_and_Tools/C/
2. Start your Windows SSH client. Set Host_Name = yourmysqlserver_URL_or_IP. Set userid=your_userid to log in to your server. This userid value might not be the same as the username of your MySQL account.

3. Set up port forwarding. Either do a remote forward (Set `local_port: 3306`, `remote_host: yourmysqlservername_or_ip`, `remote_port: 3306`) or a local forward (Set `port: 3306`, `host: localhost`, `remote port: 3306`).
4. Save everything, otherwise you will have to redo it the next time.
5. Log in to your server with the SSH session you just created.
6. On your Windows machine, start some ODBC application (such as Access).
7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in `localhost` for the MySQL host server, not *yourmysqlservername*.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.

5.10. Backup and Recovery

This section discusses how to make database backups (full and incremental) and how to perform table maintenance. The syntax of the SQL statements described here is given in [Chapter 13, SQL Statement Syntax](#). Much of the information here pertains primarily to MyISAM tables. Additional information about InnoDB backup procedures is given in [Section 14.2.8, “Backing Up and Recovering an InnoDB Database”](#).

5.10.1. Database Backups

Because MySQL tables are stored as files, it is easy to do a backup. To get a consistent backup, do a `LOCK TABLES` on the relevant tables, followed by `FLUSH TABLES` for the tables. See [Section 13.4.5, “LOCK TABLES and UNLOCK TABLES Syntax”](#), and [Section 13.5.5.2, “FLUSH Syntax”](#). You need only a read lock; this allows other clients to continue to query the tables while you are making a copy of the files in the database directory. The `FLUSH TABLES` statement is needed to ensure that the all active index pages are written to disk before you start the backup.

To make an SQL-level backup of a table, you can use `SELECT INTO ... OUTFILE`. For this statement, the output file cannot already exist because allowing files to be overwritten would constitute a security risk. See [Section 13.2.7, “SELECT Syntax”](#).

Another technique for backing up a database is to use the **mysqldump** program or the **mysqlhotcopy** script. See [Section 8.12, “mysqldump — A Database Backup Program”](#), and [Section 8.13, “mysqlhotcopy — A Database Backup Program”](#).

1. Create a full backup of your database:

```
shell> mysqldump --tab=/path/to/some/dir --opt db_name
```

Or:

```
shell> mysqlhotcopy db_name /path/to/some/dir
```

You can also create a binary backup simply by copying all table files (*.frm, *.MYD, and *.MYI files), as long as the server isn't updating anything. The **mysqlhotcopy** script uses this method. (But note that these methods do not work if your database contains InnoDB tables. InnoDB does not store table contents in database directories, and **mysqlhotcopy** works only for MyISAM tables.)

2. Stop **mysqld** if it is running, then start it with the `--log-bin[=file_name]` option. See [Section 5.12.3, “The Binary Log”](#). The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you executed **mysqldump**.

For InnoDB tables, it is possible to perform an online backup that takes no locks on tables; see [Section 8.12, “mysqldump — A Database Backup Program”](#).

MySQL supports incremental backups: You need to start the server with the `--log-bin` option to enable binary logging; see [Section 5.12.3, “The Binary Log”](#). At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using `FLUSH LOGS`. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained further below. The next time you do a full backup, you should also rotate the binary log using `FLUSH LOGS`, `mysqldump --flush-logs`, or `mysqlhotcopy --flushlog`. See [Section 8.12, “mysqldump — A Database Backup Program”](#), and [Section 8.13, “mysqlhotcopy — A Database Backup Program”](#).

If your MySQL server is a slave replication server, then regardless of the backup method you choose, you should also back up the `master.info` and `relay-log.info` files when you back up your slave's data. These files are always needed to resume replication after you restore the slave's data. If your slave is subject to replicating `LOAD DATA INFILE` commands, you should also back up any `SQL_LOAD-*` files that may exist in the directory specified by the `--slave-load-tmpdir` option. (This location defaults to the value of the `tmpdir` variable if not specified.) The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations.

If you have to restore MyISAM tables, try to recover them using `REPAIR TABLE` or

myisamchk -r first. That should work in 99.9% of all cases. If **myisamchk** fails, try the following procedure. Note that it works only if you have enabled binary logging by starting MySQL with the `--log-bin` option.

1. Restore the original **mysqldump** backup, or binary backup.
2. Execute the following command to re-run the updates in the binary logs:

```
shell> mysqlbinlog binlog.[0-9]* | mysql
```

In some cases, you may want to re-run only certain binary logs, from certain positions (usually you want to re-run all binary logs from the date of the restored backup, excepting possibly some incorrect statements). See [Section 8.10, “mysqlbinlog — Utility for Processing Binary Log Files”](#), for more information on the **mysqlbinlog** utility and how to use it.

You can also make selective backups of individual files:

- To dump the table, use `SELECT * INTO OUTFILE 'file_name' FROM tbl_name`.
- To reload the table, use `LOAD DATA INFILE 'file_name' REPLACE ...`. To avoid duplicate rows, the table must have a `PRIMARY KEY` or a `UNIQUE` index. The `REPLACE` keyword causes old rows to be replaced with new ones when a new row duplicates an old row on a unique key value.

If you have performance problems with your server while making backups, one strategy that can help is to set up replication and perform backups on the slave rather than on the master. See [Section 6.1, “Introduction to Replication”](#).

If you are using a Veritas filesystem, you can make a backup like this:

1. From a client program, execute `FLUSH TABLES WITH READ LOCK`.
2. From another shell, execute `mount vxfss snapshot`.
3. From the first client, execute `UNLOCK TABLES`.
4. Copy files from the snapshot.
5. Unmount the snapshot.

5.10.2. Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that allows you to recover data after several types of crashes:

- Operating system crash
- Power failure
- Filesystem crash
- Hardware problem (hard drive, motherboard, and so forth)

The example commands do not include options such as `--user` and `--password` for the **mysqldump** and **mysql** programs. You should include such options as necessary so that the MySQL server allows you to connect to it.

We assume that data is stored in the InnoDB storage engine, which has support for transactions and automatic crash recovery. We also assume that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The InnoDB data files might not contain consistent data due to the crash, but InnoDB reads its logs and finds in them the list of pending committed and non-committed transactions that have not been flushed to the data files. InnoDB automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
```

```
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

For the cases of filesystem crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that we must already have made backups. To make sure that is the case, we should design a backup policy.

5.10.2.1. Backup Policy

We all know that backups must be scheduled periodically. A full backups (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, InnoDB Hot Backup provides online non-blocking physical backup of the InnoDB data files, and **mysqldump** provides online logical backup. This discussion uses **mysqldump**.

Assume that we make a backup on Sunday at 1 p.m., when load is low. The following command makes a full backup of all our InnoDB tables in all databases:

```
shell> mysqldump --single-transaction --all-databases > backup_sunda
```

This is an online, non-blocking backup that does not disturb the reads and writes on the tables. We assumed earlier that our tables are InnoDB tables, so `--single-transaction` uses a consistent read and guarantees that data seen by **mysqldump** does not change. (Changes made by other clients to InnoDB tables are not seen by the **mysqldump** process.) If we do also have other types of tables, we must assume that they are not changed during the backup. For example, for the MyISAM tables in the `mysql` database, we must assume that no administrative changes are being made to MySQL accounts during the backup.

The resulting `.sql` file produced by **mysqldump** contains a set of SQL `INSERT` statements that can be used to reload the dumped tables at a later time.

Full backups are necessary, but they are not always convenient. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. After we have made the initial full backup, it is more efficient to make incremental backups. They are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. The MySQL server should always be started with the `--log-bin` option so that it stores these changes in a file while it updates data. This option enables binary logging, so that the server writes each SQL statement that updates data into a file called a MySQL binary log. Looking at the data directory of a MySQL server that was started with the `--log-bin` option and that has been running for some days, we find these MySQL binary log files:

```
-rw-rw---- 1 guilhem guilhem 1277324 Nov 10 23:59 gbichot2-bin.00
-rw-rw---- 1 guilhem guilhem      4 Nov 10 23:59 gbichot2-bin.00
-rw-rw---- 1 guilhem guilhem    79 Nov 11 11:06 gbichot2-bin.00
-rw-rw---- 1 guilhem guilhem   508 Nov 11 11:08 gbichot2-bin.00
-rw-rw---- 1 guilhem guilhem 220047446 Nov 12 16:47 gbichot2-bin.00
-rw-rw---- 1 guilhem guilhem  998412 Nov 14 10:08 gbichot2-bin.00
-rw-rw---- 1 guilhem guilhem   361 Nov 14 10:07 gbichot2-bin.in
```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a **mysqladmin flush-logs** command.

mysqldump also has an option to flush the logs. The `.index` file in the data directory contains the list of all MySQL binary logs in the directory. This file is used for replication.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, then any binary log files created afterward contain all the data changes made since the backup. Let's modify the previous **mysqldump** command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so

that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
--all-databases > backup_sunday_1_PM.sql
```

After executing this command, the data directory contains a new binary log file, `gbichot2-bin.000007`. The resulting `.sql` file includes these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG
```

Because the **mysqldump** command made a full backup, those lines mean two things:

- The `.sql` file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or newer.
- All data changes logged after the backup are not present in the `.sql`, but are present in the `gbichot2-bin.000007` binary log file or newer.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a **mysqladmin flush-logs** command creates `gbichot2-bin.000008`. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. will be in the `gbichot2-bin.000007` file. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another **mysqladmin flush-logs** command. All changes between Monday 1 p.m. and Tuesday 1 p.m. will be in the `gbichot2-bin.000008` file (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
--all-databases --delete-master-logs > backup_sunday_1_PM.s
```

Note: Deleting the MySQL binary logs with **mysqldump --delete-master-logs** can be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log. The description for the `PURGE MASTER LOGS` statement explains what should be

verified before deleting the MySQL binary logs. See [Section 13.6.1.1, “PURGE MASTER LOGS Syntax”](#).

5.10.2.2. Using Backups for Recovery

Now, suppose that we have a catastrophic crash on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the `gbichot2-bin.000007` and `gbichot2-bin.000008` binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a `--log-bin` option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the `gbichot2-bin.000009` file at hand, and we could apply it using **mysqlbinlog** and **mysql** to restore the most recent data changes with no loss up to the moment of the crash.

5.10.2.3. Backup Strategy Summary

In case of an operating system crash or power failure, InnoDB itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the `--log-bin` option, or even `--log-`

`bin=log_name`, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).

- Make periodic full backups, using the **mysqldump** command shown earlier in [Section 5.10.2.1, “Backup Policy”](#), that makes an online, non-blocking backup.
- Make periodic incremental backups by flushing the logs with `FLUSH LOGS` or **mysqladmin flush-logs**.

5.10.3. Point-in-Time Recovery

If a MySQL server was started with the `--log-bin` option to enable binary logging, you can use the **mysqlbinlog** utility to recover data from the binary log files, starting from a specified point in time (for example, since your last backup) until the present or another specified point in time. For information on enabling the binary log and using **mysqlbinlog**, see [Section 5.12.3, “The Binary Log”](#), and [Section 8.10, “mysqlbinlog — Utility for Processing Binary Log Files”](#).

To restore data from a binary log, you must know the location and name of the current binary log file. By default, the server creates binary log files in the data directory, but a pathname can be specified with the `--log-bin` option to place the files in a different location. Typically the option is given in an option file (that is, `my.cnf` or `my.ini`, depending on your system). It can also be given on the command line when the server is started. To determine the name of the current binary log file, issue the following statement:

```
mysql> SHOW BINLOG EVENTS\G
```

If you prefer, you can execute the following command from the command line instead:

```
shell> mysql -u root -p -E -e "SHOW BINLOG EVENTS"
```

Enter the root password for your server when **mysql** prompts you for it.

5.10.3.1. Specifying Times for Recovery

To indicate the start and end times for recovery, specify the `--start-date` and `--stop-date` options for **mysqlbinlog**, in DATETIME format. As an example, suppose that exactly at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

```
shell> mysqlbinlog --stop-date="2005-04-20 9:59:59" \  
        /var/log/mysql/bin.123456 | mysql -u root -p
```

This command recovers all of the data up until the date and time given by the `--stop-date` option. If you did not detect the erroneous SQL statement that was entered until hours later, you will probably also want to recover the activity that occurred afterward. Based on this, you could run **mysqlbinlog** again with a start date and time, like so:

```
shell> mysqlbinlog --start-date="2005-04-20 10:01:00" \  
        /var/log/mysql/bin.123456 | mysql -u root -p
```

In this command, the SQL statements logged from 10:01 a.m. on will be re-executed. The combination of restoring of the previous night's dump file and the two **mysqlbinlog** commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on. You should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

Then open the file with a text editor to examine it.

5.10.3.2. Specifying Positions for Recovery

Instead of specifying dates and times, the `--start-position` and `--stop-position` options for **mysqlbinlog** can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions occurred around the same time as a damaging SQL statement. To determine the position numbers, run **mysqlbinlog** for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

```
shell> mysqlbinlog --start-date="2005-04-20 9:55:00" \  
--stop-date="2005-04-20 10:05:00" \  
/var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

This command creates a small text file in the /tmp directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you don't want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as log_pos followed by a number. After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

```
shell> mysqlbinlog --stop-position="368312" /var/log/mysql/bin.12345  
| mysql -u root -p
```

```
shell> mysqlbinlog --start-position="368315" /var/log/mysql/bin.1234  
| mysql -u root -p
```

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of **mysqlbinlog** includes SET TIMESTAMP statements before each SQL statement recorded, the recovered data and related MySQL logs will reflect the original times at which the transactions were executed.

5.10.4. Table Maintenance and Crash Recovery

This section discusses how to use **myisamchk** to check or repair MyISAM tables (tables that have .MYD and .MYI files for storing data and indexes). For general **myisamchk** background, see [Section 8.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

You can use **myisamchk** to get information about your database tables or to check, repair, or optimize them. The following sections describe how to perform these operations and how to set up a table maintenance schedule.

Even though table repair with **myisamchk** is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table

myisamchk operations that affect indexes can cause FULLTEXT indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in [Section 8.3.1, “myisamchk General Options”](#).

In many cases, you may find it simpler to do MyISAM table maintenance using the SQL statements that perform operations that **myisamchk** can do:

- To check or repair MyISAM tables, use CHECK TABLE or REPAIR TABLE.
- To optimize MyISAM tables, use OPTIMIZE TABLE.
- To analyze MyISAM tables, use ANALYZE TABLE.

These statements can be used directly or by means of the **mysqlcheck** client program. One advantage of these statements over **myisamchk** is that the server does all the work. With **myisamchk**, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between **myisamchk** and the server. See [Section 13.5.2.1, “ANALYZE TABLE Syntax”](#), [Section 13.5.2.3, “CHECK TABLE Syntax”](#), [Section 13.5.2.5, “OPTIMIZE TABLE Syntax”](#), and [Section 13.5.2.6, “REPAIR TABLE Syntax”](#).

5.10.4.1. Using myisamchk for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#).

For an explanation of how MyISAM tables can become corrupted, see [Section 14.1.4, “MyISAM Table Problems”](#).

If you run **mysqld** with external locking disabled (which is the default as of MySQL 4.0), you cannot reliably use **myisamchk** to check a table when **mysqld** is using the same table. If you can be certain that no one will access the tables through **mysqld** while you run **myisamchk**, you only have to execute **mysqladmin flush-tables** before you start checking the tables. If you cannot guarantee this, you must stop **mysqld** while you check the tables. If you run **myisamchk** to check tables that **mysqld** is updating at the same time, you may get a warning that a table is corrupt even when it is not.

If the server is run with external locking enabled, you can use **myisamchk** to check tables at any time. In this case, if the server tries to update a table that **myisamchk** is using, the server will wait for **myisamchk** to finish before it continues.

If you use **myisamchk** to repair or optimize tables, you *must* always ensure that the **mysqld** server is not using the table (this also applies if external locking is disabled). If you don't stop **mysqld**, you should at least do a **mysqladmin flush-tables** before you run **myisamchk**. Your tables *may become corrupted* if the server and **myisamchk** access the tables simultaneously.

When performing crash recovery, it is important to understand that each MyISAM table *tbl_name* in a database corresponds to three files in the database directory:

File	Purpose
tbl_name.frm	Definition (format) file
tbl_name.MYD	Data file
tbl_name.MYI	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

myisamchk works by creating a copy of the .MYD data file row by row. It ends the repair stage by removing the old .MYD file and renaming the new file to the original file name. If you use **--quick**, **myisamchk** does not create a temporary .MYD file, but instead assumes that the .MYD file is correct and generates only a new index file without touching the .MYD file. This is safe, because **myisamchk** automatically detects whether the .MYD file is corrupt and aborts the repair if it is. You can also specify the **--quick** option twice to **myisamchk**. In this case, **myisamchk** does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the .MYD file. Normally the use of two **--quick** options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running **myisamchk**.

5.10.4.2. How to Check MyISAM Tables for Errors

To check a MyISAM table, use the following commands:

- `myisamchk tbl_name`

This finds 99.99% of all errors. What it cannot find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run **myisamchk** without options or with the `-s` (silent) option.

- `myisamchk -m tbl_name`

This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- `myisamchk -e tbl_name`

This does a complete and thorough check of all data (`-e` means “extended check”). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, **myisamchk** stops after the first error it finds. If you want to obtain more information, you can add the `-v` (verbose) option. This causes **myisamchk** to keep going, up through a maximum of 20 errors.

- `myisamchk -e -i tbl_name`

This is like the previous command, but the `-i` option tells **myisamchk** to print additional statistical information.

In most cases, a simple **myisamchk** command with no arguments other than the table name is sufficient to check a table.

5.10.4.3. How to Repair Tables

The discussion in this section describes how to use **myisamchk** on MyISAM tables (extensions `.MYI` and `.MYD`).

You can also (and should, if possible) use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair MyISAM tables. See [Section 13.5.2.3, “CHECK](#)

[TABLE Syntax](#)”, and [Section 13.5.2.6, “REPAIR TABLE Syntax”](#).”

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

- `tbl_name.frm` is locked against change
- Can't find file `tbl_name.MYI` (Errcode: *nnn*)
- Unexpected end of file
- Record file is crashed
- Got error *nnn* from table handler

To get more information about the error, run **pererror** *nnn*, where *nnn* is the error number. The following example shows how to use **pererror** to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> pererror 126 127 132 134 135 136 141 144 145
126 = Index file is crashed / Wrong file format
127 = Record-file is crashed
132 = Old database file
134 = Record was already deleted (or record file crashed)
135 = No more room in record file
136 = No more room in index file
141 = Duplicate unique key or constraint on write or update
144 = Table is crashed and last repair failed
145 = Table was marked as crashed and should be repaired
```

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use `ALTER TABLE` to increase the `MAX_ROWS` and `AVG_ROW_LENGTH` table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you do not know the current table option values, use `SHOW CREATE TABLE`.

For the other errors, you must repair your tables. **myisamchk** can usually detect and fix most problems that occur.

The repair process involves up to four stages, described here. Before you begin,

you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that **mysqld** runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This section is for the cases where a table check fails (such as those described in [Section 5.10.4.2, “How to Check MyISAM Tables for Errors”](#)), or you want to use the extended features that **myisamchk** provides.

The options that you can use for table maintenance with **myisamchk** are described in [Section 8.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

If you are going to repair a table from the command line, you must first stop the **mysqld** server. Note that when you do **mysqladmin shutdown** on a remote server, the **mysqld** server is still alive for a while after **mysqladmin** returns, until all statement-processing has stopped and all index changes have been flushed to disk.

Stage 1: Checking your tables

Run **myisamchk *.MYI** or **myisamchk -e *.MYI** if you have more time. Use the **-s** (silent) option to suppress unnecessary information.

If the **mysqld** server is stopped, you should use the **--update-state** option to tell **myisamchk** to mark the table as “checked.”

You have to repair only those tables for which **myisamchk** announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as out of memory errors), or if **myisamchk** crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try **myisamchk -r -q tbl_name** (**-r -q** means “quick recovery mode”). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use **myisamchk -r tbl_name** (-r means “recovery mode”). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.
3. If the preceding step fails, use **myisamchk --safe-recover tbl_name**. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).

Note: If you want a repair operation to go much faster, you should set the values of the `sort_buffer_size` and `key_buffer_size` variables each to about 25% of your available memory when running **myisamchk**.

If you get unexpected errors when repairing (such as out of memory errors), or if **myisamchk** crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.
2. Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
mysql> SET AUTOCOMMIT=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

3. Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)

Go back to Stage 2. **myisamchk -r -q** should work. (This should not be an endless loop.)

You can also use the `REPAIR TABLE tbl_name USE_FRM SQL` statement, which performs the whole procedure automatically. There is also no possibility of

unwanted interaction between a utility and the server, because the server does all the work when you use `REPAIR TABLE`. See [Section 13.5.2.6, “REPAIR TABLE Syntax”](#).

Stage 4: Very difficult repair

You should reach this stage only if the `.frm` description file has also crashed. That should never happen, because the description file is not changed after the table is created:

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with **`myisamchk -r`**.
2. If you do not have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, and then move the `.frm` description and `.MYI` index files from the other database to your crashed database. This gives you new description and index files, but leaves the `.MYD` data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

5.10.4.4. Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run **`myisamchk`** in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the `OPTIMIZE TABLE SQL` statement. `OPTIMIZE TABLE` does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See [Section 13.5.2.5, “OPTIMIZE TABLE Syntax”](#).

`myisamchk` has a number of other options that you can use to improve the performance of a table:

- `--analyze, -a`

- `--sort-index, -S`
- `--sort-records=index_num, -R index_num`

For a full description of all available options, see [Section 8.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

5.10.4.5. Getting Information About a Table

To obtain a description of a table or statistics about it, use the commands shown here. We explain some of the information in more detail later.

- **`myisamchk -d tbl_name`**

Runs **`myisamchk`** in “describe mode” to produce a description of your table. If you start the MySQL server with external locking disabled, **`myisamchk`** may report an error for a table that is updated while it runs. However, because **`myisamchk`** does not change the table in describe mode, there is no risk of destroying data.

- **`myisamchk -d -v tbl_name`**

Adding `-v` runs **`myisamchk`** in verbose mode so that it produces more information about what it is doing.

- **`myisamchk -eis tbl_name`**

Shows only the most important information from a table. This operation is slow because it must read the entire table.

- **`myisamchk -eiv tbl_name`**

This is like `-eis`, but tells you what is being done.

Sample output for some of these commands follows. They are based on a table with these data and index file sizes:

```
-rw-rw-r--  1 monty    tcx      317235748 Jan 12 17:30 company.MYD
-rw-rw-r--  1 davida   tcx      96482304  Jan 12 18:35 company.MYI
```

Example of **myisamchk -d** output:

MyISAM file: company.MYI
Record format: Fixed length
Data records: 1403698 Deleted blocks: 0
Recordlength: 226

table description:

Key	Start	Len	Index	Type
1	2	8	unique	double
2	15	10	multip.	text packed stripped
3	219	8	multip.	double
4	63	10	multip.	text packed stripped
5	167	2	multip.	unsigned short
6	177	4	multip.	unsigned long
7	155	4	multip.	text
8	138	4	multip.	unsigned long
9	177	4	multip.	unsigned long
	193	1		text

Example of **myisamchk -d -v** output:

MyISAM file: company
Record format: Fixed length
File-version: 1
Creation time: 1999-10-30 12:12:51
Recover time: 1999-10-31 19:13:01
Status: checked
Data records: 1403698 Deleted blocks: 0
Datafile parts: 1403698 Deleted data: 0
Datafile pointer (bytes): 3 Keyfile pointer (bytes): 3
Max datafile length: 3791650815 Max keyfile length: 4294967294
Recordlength: 226

table description:

Key	Start	Len	Index	Type	Rec/key	Root	Blocksi
1	2	8	unique	double	1	15845376	10
2	15	10	multip.	text packed stripped	2	25062400	10
3	219	8	multip.	double	73	40907776	10
4	63	10	multip.	text packed stripped	5	48097280	10
5	167	2	multip.	unsigned short	4840	55200768	10
6	177	4	multip.	unsigned long	1346	65145856	10
7	155	4	multip.	text	4995	75090944	10
8	138	4	multip.	unsigned long	87	85036032	10
9	177	4	multip.	unsigned long	178	96481280	10
	193	1		text			

Example of **myisamchk -eis** output:

Checking MyISAM file: company

Key: 1:	Keyblocks used:	97%	Packed:	0%	Max levels:	4
Key: 2:	Keyblocks used:	98%	Packed:	50%	Max levels:	4
Key: 3:	Keyblocks used:	97%	Packed:	0%	Max levels:	4
Key: 4:	Keyblocks used:	99%	Packed:	60%	Max levels:	3
Key: 5:	Keyblocks used:	99%	Packed:	0%	Max levels:	3
Key: 6:	Keyblocks used:	99%	Packed:	0%	Max levels:	3
Key: 7:	Keyblocks used:	99%	Packed:	0%	Max levels:	3
Key: 8:	Keyblocks used:	99%	Packed:	0%	Max levels:	3
Key: 9:	Keyblocks used:	98%	Packed:	0%	Max levels:	4
Total:	Keyblocks used:	98%	Packed:	17%		

Records:	1403698	M.recordlength:	226
Packed:	0%		
Recordspace used:	100%	Empty space:	0%
Blocks/Record:	1.00		
Record blocks:	1403698	Delete blocks:	0
Recorddata:	317235748	Deleted data:	0
Lost space:	0	Linkdata:	0

User time 1626.51, System time 232.36

Maximum resident set size 0, Integral resident set size 0

Non physical pagefaults 0, Physical pagefaults 627, Swaps 0

Blocks in 0 out 0, Messages in 0 out 0, Signals 0

Voluntary context switches 639, Involuntary context switches 28966

Example of **myisamchk -eiv** output:

Checking MyISAM file: company

Data records: 1403698 Deleted blocks: 0

- check file-size

- check delete-chain

block_size 1024:

index 1:

index 2:

index 3:

index 4:

index 5:

index 6:

index 7:

index 8:

index 9:

No recordlinks

- check index reference

- check data record references index: 1

Key: 1:	Keyblocks used:	97%	Packed:	0%	Max levels:	4
---------	-----------------	-----	---------	----	-------------	---

- check data record references index: 2

Key: 2:	Keyblocks used:	98%	Packed:	50%	Max levels:	4
---------	-----------------	-----	---------	-----	-------------	---

- check data record references index: 3

```

Key: 3: Keyblocks used: 97% Packed: 0% Max levels: 4
- check data record references index: 4
Key: 4: Keyblocks used: 99% Packed: 60% Max levels: 3
- check data record references index: 5
Key: 5: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 6
Key: 6: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 7
Key: 7: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 8
Key: 8: Keyblocks used: 99% Packed: 0% Max levels: 3
- check data record references index: 9
Key: 9: Keyblocks used: 98% Packed: 0% Max levels: 4
Total: Keyblocks used: 9% Packed: 17%

```

- check records and index references
**** LOTS OF ROW NUMBERS DELETED ****

```

Records:          1403698  M.recordlength:  226  Packed:
Recordspace used:   100%  Empty space:      0%  Blocks/Record:  1.
Record blocks:     1403698  Delete blocks:    0
Recorddata:        317235748  Deleted data:    0
Lost space:         0  Linkdata:        0

```

User time 1639.63, System time 251.61
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 10580, Swaps 0
Blocks in 4 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 10604, Involuntary context switches 12279

Explanations for the types of information **myisamchk** produces are given here.
“Keyfile” refers to the index file. “Record” and “row” are synonymous.

- MyISAM file

Name of the MyISAM (index) file.

- File-version

Version of MyISAM format. Currently always 2.

- Creation time

When the data file was created.

- Recover time

When the index/data file was last reconstructed.

- Data records

How many rows are in the table.

- Deleted blocks

How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See [Section 5.10.4.4, “Table Optimization”](#).

- Datafile parts

For dynamic-row format, this indicates how many data blocks there are. For an optimized table without fragmented rows, this is the same as Data records.

- Deleted data

How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See [Section 5.10.4.4, “Table Optimization”](#).

- Datafile pointer

The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a row address. For dynamic tables, this is a byte address.

- Keyfile pointer

The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- Max datafile length

How long the table data file can become, in bytes.

- Max keyfile length

How long the table index file can become, in bytes.

- Recordlength

How much space each row takes, in bytes.

- Record format

The format used to store table rows. The preceding examples use Fixed length. Other possible values are Compressed and Packed.

- table description

A list of all keys in the table. For each key, **myisamchk** displays some low-level information:

- Key

This key's number.

- Start

Where in the row this portion of the index starts.

- Len

How long this portion of the index is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column.

- Index

Whether a key value can exist multiple times in the index. Possible values are unique or multip. (multiple).

- Type

What data type this portion of the index has. This is a MyISAM data type with the possible values packed, stripped, or empty.

- Root

Address of the root index block.

- Blocksize

The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

- Rec/key

This is a statistical value used by the optimizer. It tells how many rows there are per value for this index. A unique index always has a value of 1. This may be updated after a table is loaded (or greatly changed) with **myisamchk -a**. If this is not updated at all, a default value of 30 is given.

For the table shown in the examples, there are two table description lines for the ninth index. This indicates that it is a multiple-part index with two parts.

- Keyblocks used

What percentage of the keyblocks are used. When a table has just been reorganized with **myisamchk**, as for the table in the examples, the values are very high (very near the theoretical maximum).

- Packed

MySQL tries to pack key values that have a common suffix. This can only be used for indexes on CHAR and VARCHAR columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the third of the preceding examples, the fourth key is 10 characters long and a 60% reduction in space is achieved.

- Max levels

How deep the B-tree for this key is. Large tables with long key values get high values.

- Records

How many rows are in the table.

- M.recordlength

The average row length. This is the exact row length for tables with fixed-length rows, because all rows have the same length.

- Packed

MySQL strips spaces from the end of strings. The Packed value indicates the percentage of savings achieved by doing this.

- Recordspace used

What percentage of the data file is used.

- Empty space

What percentage of the data file is unused.

- Blocks/Record

Average number of blocks per row (that is, how many links a fragmented row is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too large, you can reorganize the table. See [Section 5.10.4.4, “Table Optimization”](#).

- Recordblocks

How many blocks (links) are used. For fixed-format tables, this is the same as the number of rows.

- Deleteblocks

How many blocks (links) are deleted.

- Recorddata

How many bytes in the data file are used.

- Deleted data

How many bytes in the data file are deleted (unused).

- Lost space

If a row is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- Linkdata

When the dynamic table format is used, row fragments are linked with pointers (4 to 7 bytes each). Linkdata is the sum of the amount of storage used by all such pointers.

If a table has been compressed with **myisampack**, **myisamchk -d** prints additional information about each table column. See [Section 8.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#), for an example of this information and a description of what it means.

5.10.4.6. Setting Up a Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair MyISAM tables is with the `CHECK TABLE` and `REPAIR TABLE` statements. See [Section 13.5.2.3, “CHECK TABLE Syntax”](#), and [Section 13.5.2.6, “REPAIR TABLE Syntax”](#).

Another way to check tables is to use **myisamchk**. For maintenance purposes, you can use **myisamchk -s**. The `-s` option (short for `--silent`) causes **myisamchk** to run in silent mode, printing messages only when errors occur.

It is also a good idea to enable automatic MyISAM table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are “expected crashed tables.”) To check MyISAM tables automatically, start the server with the `--myisam-recover` option. See [Section 5.2.1, “mysqld Command Options”](#).

You should also check your tables regularly during normal system operation. At

MySQL AB, we run a **cron** job to check all our important tables once a week, using a line like this in a crontab file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.*M
```

This prints out information about crashed tables so that we can examine and repair them when needed.

Because we have not had any unexpectedly crashed tables (tables that become corrupted for reasons other than hardware trouble) for several years, once a week is more than sufficient for us.

We recommend that to start with, you execute **myisamchk -s** each night on all tables that have been updated during the last 24 hours, until you come to trust MySQL as much as we do.

Normally, MySQL tables need little maintenance. If you are performing many updates to MyISAM tables with dynamic-sized rows (tables with VARCHAR, BLOB, or TEXT columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using `OPTIMIZE TABLE` on the tables in question. Alternatively, if you can stop the **mysqld** server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index --sort_buffer_size=16M /*/*.MYI
```

5.11. MySQL Localization and International Usage

This section describes how to configure the server to use different character sets. It also discusses how to set the server's time zone and enable per-connection time zone support.

5.11.1. The Character Set Used for Data and Sorting

By default, MySQL uses the `latin1` (cp1252 West European) character set and the `latin1_swedish_ci` collation that sorts according to Swedish/Finnish rules. These defaults are suitable for the United States and most of Western Europe.

All MySQL binary distributions are compiled with `--with-extra-charsets=complex`. This adds code to all standard programs that enables them to handle `latin1` and all multi-byte character sets within the binary. Other character sets are loaded from a character-set definition file when needed.

The character set determines what characters are allowed in identifiers. The collation determines how strings are sorted by the `ORDER BY` and `GROUP BY` clauses of the `SELECT` statement.

You can change the default server character set and collation with the `--character-set-server` and `--collation-server` options when you start the server. The collation must be a legal collation for the default character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.) See [Section 5.2.1, “mysqld Command Options”](#).

The character sets available depend on the `--with-charset=charset_name` and `--with-extra-charsets=list-of-charsets | complex | all | none` options to **configure**, and the character set configuration files listed in `SHAREDIR/charsets/Index`. See [Section 2.9.2, “Typical configure Options”](#).

If you change the character set when running MySQL, that may also change the sort order. Consequently, you must run **`myisamchk -r -q --set-collation=collation_name`** on all MyISAM tables, or your indexes may not be ordered correctly.

When a client connects to a MySQL server, the server indicates to the client

what the server's default character set is. The client switches to this character set for this connection.

You should use `mysql_real_escape_string()` when escaping strings for an SQL query. `mysql_real_escape_string()` is identical to the old `mysql_escape_string()` function, except that it takes the MySQL connection handle as the first parameter so that the appropriate character set can be taken into account when escaping characters.

If the client is compiled with paths that differ from where the server is installed and the user who configured MySQL didn't include all character sets in the MySQL binary, you must tell the client where it can find the additional character sets it needs if the server runs with a different character set from the client. You can do this by specifying a `--character-sets-dir` option to indicate the path to the directory in which the dynamic MySQL character sets are stored. For example, you can put the following in an option file:

```
[client]
character-sets-dir=/usr/local/mysql/share/mysqlCharsets
```

You can force the client to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This is normally unnecessary, however.

5.11.1.1. Using the German Character Set

In MySQL 5.0, character set and collation are specified separately. This means that if you want German sort order, you should select the `latin1` character set and either the `latin1_german1_ci` or `latin1_german2_ci` collation. For example, to start the server with the `latin1_german1_ci` collation, use the `--character-set-server=latin1` and `--collation-server=latin1_german1_ci` options.

For information on the differences between these two collations, see [Section 10.9.2, “West European Character Sets”](#).

5.11.2. Setting the Error Message Language

By default, **mysqld** produces error messages in English, but they can also be displayed in any of these other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish.

To start **mysqld** with a particular language for error messages, use the `--language` or `-L` option. The option value can be a language name or the full path to the error message file. For example:

```
shell> mysqld --language=swedish
```

Or:

```
shell> mysqld --language=/usr/local/share/swedish
```

The language name should be specified in lowercase.

By default, the language files are located in the `share/LANGUAGE` directory under the MySQL base directory.

You can also change the content of the error messages produced by the server. Details can be found in the MySQL Internals manual, available at <http://dev.mysql.com/doc/>. If you upgrade to a newer version of MySQL after changing the error messages, remember to repeat your changes after the upgrade.

5.11.3. Adding a New Character Set

This section discusses the procedure for adding a new character set to MySQL. You must have a MySQL source distribution to use these instructions. To choose the proper procedure, determine whether the character set is simple or complex:

- If the character set does not need to use special string collating routines for sorting and does not need multi-byte character support, it is simple.
- If it needs either of those features, it is complex.

For example, `latin1` and `danish` are simple character sets, whereas `big5` and `czech` are complex character sets.

In the following instructions, the name of the character set is represented by *MYSET*.

For a simple character set, do the following:

1. Add *MYSET* to the end of the `sql/share/charsets/Index` file. Assign a unique number to it.
2. Create the file `sql/share/charsets/MYSET.conf`. (You can use a copy of `sql/share/charsets/latin1.conf` as the basis for this file.)

The syntax for the file is very simple:

- Comments start with a '#' character and continue to the end of the line.
- Words are separated by arbitrary amounts of whitespace.
- When defining the character set, every word must be a number in hexadecimal format.
- The `ctype` array takes up the first 257 words. The `to_lower[]`, `to_upper[]` and `sort_order[]` arrays take up 256 words each after that.

See [Section 5.11.4, “The Character Definition Arrays”](#).

3. Add the character set name to the `CHARSETS_AVAILABLE` and `COMPILED_CHARSETS` lists in `configure.in`.
4. Reconfigure, recompile, and test.

For a complex character set, do the following:

1. Create the file `strings/ctype-MYSET.c` in the MySQL source distribution.
2. Add *MYSET* to the end of the `sql/share/charsets/Index` file. Assign a unique number to it.
3. Look at one of the existing `ctype-*.c` files (such as `strings/ctype-big5.c`) to see what needs to be defined. Note that the arrays in your file must have names like `ctype_MYSET`, `to_lower_MYSET`, and so on. These

correspond to the arrays for a simple character set. See [Section 5.11.4, “The Character Definition Arrays”](#).

4. Near the top of the file, place a special comment like this:

```
/*  
 * This comment is parsed by configure to create ctype.c,  
 * so don't change it unless you know what you are doing.  
 *  
 * .configure. number_MYSET=MYNUMBER  
 * .configure. strxfrm_multiply_MYSET=N  
 * .configure. mbmaxlen_MYSET=N  
 */
```

The **configure** program uses this comment to include the character set into the MySQL library automatically.

The `strxfrm_multiply` and `mbmaxlen` lines are explained in the following sections. You need include them only if you need the string collating functions or the multi-byte character set functions, respectively.

5. You should then create some of the following functions:

- `my_strncoll_MYSET()`
- `my_strcoll_MYSET()`
- `my_strxfrm_MYSET()`
- `my_like_range_MYSET()`

See [Section 5.11.5, “String Collating Support”](#).

6. Add the character set name to the `CHARSETS_AVAILABLE` and `COMPILED_CHARSETS` lists in `configure.in`.

7. Reconfigure, recompile, and test.

The `sql/share/charsets/README` file includes additional instructions.

If you want to have the character set included in the MySQL distribution, mail a patch to the MySQL internals mailing list. See [Section 1.7.1, “MySQL](#)

[Mailing Lists](#)".

5.11.4. The Character Definition Arrays

`to_lower[]` and `to_upper[]` are simple arrays that hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
to_lower['A'] should contain 'a'  
to_upper['a'] should contain 'A'
```

`sort_order[]` is a map indicating how characters should be ordered for comparison and sorting purposes. Quite often (but not for all character sets) this is the same as `to_upper[]`, which means that sorting is case-insensitive. MySQL sorts characters based on the values of `sort_order[]` elements. For more complicated sorting rules, see the discussion of string collating in [Section 5.11.5, "String Collating Support"](#).

`ctype[]` is an array of bit values, with one element for one character. (Note that `to_lower[]`, `to_upper[]`, and `sort_order[]` are indexed by character value, but `ctype[]` is indexed by character value + 1. This is an old legacy convention for handling EOF.)

You can find the following bitmask definitions in `m_ctype.h`:

```
#define _U      01      /* Uppercase */  
#define _L      02      /* Lowercase */  
#define _N      04      /* Numeral (digit) */  
#define _S      010     /* Spacing character */  
#define _P      020     /* Punctuation */  
#define _C      040     /* Control character */  
#define _B      0100    /* Blank */  
#define _X      0200    /* hexadecimal digit */
```

The `ctype[]` entry for each character should be the union of the applicable bitmask values that describe the character. For example, 'A' is an uppercase character (`_U`) as well as a hexadecimal digit (`_X`), so `ctype['A'+1]` should contain the value:

```
_U + _X = 01 + 0200 = 0201
```

5.11.5. String Collating Support

If the sorting rules for your language are too complex to be handled with the simple `sort_order[]` table, you need to use the string collating functions.

The best documentation for this is the existing character sets. Look at the `big5`, `czech`, `gbk`, `sjis`, and `tis160` character sets for examples.

You must specify the `strxfrm_multiply_MYSET=N` value in the special comment at the top of the file. *N* should be set to the maximum ratio the strings may grow during `my_strxfrm_MYSET` (it must be a positive integer).

5.11.6. Multi-Byte Character Support

If you want to add support for a new character set that includes multi-byte characters, you need to use the multi-byte character functions.

The best documentation for this is the existing character sets. Look at the `euc_kr`, `gb2312`, `gbk`, `sjis`, and `ujis` character sets for examples. These are implemented in the `ctype-charset_name.c` files in the `strings` directory.

You must specify the `mbmaxlen_MYSET=N` value in the special comment at the top of the source file. *N* should be set to the size in bytes of the largest character in the set.

5.11.7. Problems With Character Sets

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- Your program uses an incorrect path to determine where the character sets are stored. (Default `/usr/local/mysql/share/mysql/charsets`). This can be fixed by using the `--character-sets-dir` option when you run the program in question.
- The character set is a multi-byte character set that cannot be loaded dynamically. In this case, you must recompile the program with support for the character set.
- The character set is a dynamic character set, but you do not have a configure file for it. In this case, you should install the configure file for the

character set from a new MySQL distribution.

- If your `Index` file does not contain the name for the character set, your program displays the following error message:

```
ERROR 1105: File '/usr/local/share/mysql/charsets/?.conf'  
not found (Errcode: 2)
```

In this case, you should either get a new `Index` file or manually add the name of any missing character sets to the current file.

For `MyISAM` tables, you can check the character set name and number for a table with `myisamchk -dvv tbl_name`.

5.11.8. MySQL Server Time Zone Support

The MySQL server maintains several time zone settings:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` system variable. The value does not change thereafter.
- The server's current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial value for `time_zone` is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone. The initial value can be specified explicitly with the `--default-time-zone=timezone` option. If you have the `SUPER` privilege, you can set the global value at runtime with this statement:

```
mysql> SET GLOBAL time_zone = timezone;
```

- Per-connection time zones. Each client that connects has its own time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

```
mysql> SET time_zone = timezone;
```

The current values of the global and client-specific time zones can be retrieved like this:

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

timezone values can be given as strings indicating an offset from UTC, such as '+10:00' or '-6:00'. If the time zone information tables in the `mysql` database have been created and populated, you can also use named time zones, such as 'Europe/Helsinki', 'US/Eastern', or 'MET'. The value 'SYSTEM' can be used to indicate that the time zone should be the same as the system time zone. Time zone names are not case sensitive.

The MySQL installation procedure creates the time zone tables in the `mysql` database, but does not load them. You must do so manually. (If you are upgrading to MySQL 4.1.3 or later from an earlier version, you should create the tables by upgrading your `mysql` database. Use the instructions in [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).)

If your system has its own *zoneinfo* database (the set of files describing time zones), you should use the `mysql_tzinfo_to_sql` program for filling the time zone tables. Examples of such systems are Linux, FreeBSD, Sun Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a *zoneinfo* database, you can use the downloadable package described later in this section.

The `mysql_tzinfo_to_sql` program is used to load the time zone tables. On the command line, pass the *zoneinfo* directory pathname to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file, and to generate leap second information:

- To load a single time zone file *tz_file* that corresponds to a time zone name *tz_name*, invoke `mysql_tzinfo_to_sql` like this:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

- If your time zone needs to account for leap seconds, initialize the leap

second information like this, where *tz_file* is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

If your system doesn't have a zoneinfo database (for example, Windows or HP-UX), you can use the package of pre-built time zone tables that is available for download at <http://dev.mysql.com/downloads/timezones.html>. This package contains .frm, .MYD, and .MYI files for the MyISAM time zone tables. These tables should be part of the mysql database, so you should place the files in the mysql subdirectory of your MySQL server's data directory. The server should be stopped while you do this.

Warning: Please don't use the downloadable package if your system has a zoneinfo database. Use the **mysql_tzinfo_to_sql** utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

For information about time zone settings in replication setup, please see [Section 6.7, “Replication Features and Known Problems”](#).

5.12. MySQL Server Logs

MySQL has several different log files that can help you find out what is going on inside **mysqld**:

Log Type	Information Written to Log
The error log	Problems encountered starting, running, or stopping mysqld
The general query log	Established client connections and statements received from clients
The binary log	All statements that change data (also used for replication)
The slow log	All queries that took more than <code>long_query_time</code> seconds to execute or didn't use indexes

By default, all log files are created in the **mysqld** data directory. You can force **mysqld** to close and reopen the log files (or in some cases switch to a new log) by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement or execute **mysqladmin flush-logs** or **mysqladmin refresh**. See [Section 13.5.5.2, “FLUSH Syntax”](#).

If you are using MySQL replication capabilities, slave replication servers maintain additional log files called relay logs. These are discussed in [Chapter 6, Replication](#).

5.12.1. The Error Log

The error log file contains information indicating when **mysqld** was started and stopped and also any critical errors that occur while the server is running. If **mysqld** notices a table that needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if **mysqld** dies. The trace can be used to determine where **mysqld** died. See [Section E.1.4, “Using a Stack Trace”](#).

If **mysqld** dies unexpectedly and **mysqld_safe** needs to restart it, **mysqld_safe** writes a restarted `mysqld` message to the error log.

You can specify where **mysqld** stores the error log file with the `--log-error[=file_name]` option. If no *file_name* value is given, **mysqld** uses the name `host_name.err` and writes the file in the data directory. If you execute `FLUSH LOGS`, the error log is renamed with the suffix `-old` and **mysqld** creates a new empty log file. (No renaming occurs if the `--log-error` option was not given.)

If you do not specify `--log-error`, or (on Windows) if you use the `--console` option, errors are written to `stderr`, the standard error output. Usually this is your terminal.

On Windows, error output is always written to the `.err` file if `--console` is not given.

5.12.2. The General Query Log

The general query log is a general record of what **mysqld** is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to **mysqld**.

mysqld writes statements to the query log in the order that it receives them. This may be different from the order in which they are executed. This is in contrast to the the binary log, for which statements are written after they are executed, but before any locks are released. (Also, the query log contains all statements, whereas the binary log does not contain statements that only select data.)

To enable the general query log, start **mysqld** with the `--log[=file_name]` or `-l[file_name]` option. If no *file_name* value is given, the default name is `host_name.log` in the data directory.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). On Unix, you can rename the file and create a new one by using the following commands:

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> cp host_name-old.log backup-directory
shell> rm host_name-old.log
```

On Windows, you cannot rename the log file while the server has it open. You must stop the server and rename the file, and then restart the server to create a new log file.

5.12.3. The Binary Log

The binary log contains all statements that update data or potentially could have updated it (for example, a `DELETE` which matched no rows). Statements are stored in the form of “events” that describe the modifications. The binary log also contains information about how long each statement took that updated data.

Note: The binary log has replaced the old update log, which is no longer available as of MySQL 5.0. The binary log contains all information that is available in the update log in a more efficient format and in a manner that is transaction-safe. If you are using transactions, you must use the MySQL binary log for backups instead of the old update log.

The binary log does not contain statements that do not modify any data. If you want to log all statements (for example, to identify a problem query), use the general query log. See [Section 5.12.2, “The General Query Log”](#).

The primary purpose of the binary log is to be able to update databases during a restore operation as fully as possible, because the binary log contains all updates done after a backup was made. The binary log is also used on master replication servers as a record of the statements to be sent to slave servers. See [Chapter 6, Replication](#).

Running the server with the binary log enabled makes performance about 1% slower. However, the benefits of the binary log for restore operations and in allowing you to set up replication generally outweigh this minor performance decrement.

When started with the `--log-bin[=base_name]` option, **mysqld** writes a log file containing all SQL commands that update data. If no *base_name* value is given, the default name is the name of the host machine followed by `-bin`. If the *base_name* is given, but not as an absolute pathname, the server writes the file in the data directory. It is recommended that you specify a *base_name*; see [Section A.8.1, “Open Issues in MySQL”](#), for the reason.

If you supply an extension in the log name (for example, `--log-bin=base_name.extension`), the extension is silently removed and ignored.

mysqld appends a numeric extension to the binary log basename. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new binary log file each time it starts or flushes the logs. The server also creates a new binary log file automatically when the current log's size reaches `max_binlog_size`. A binary log file may become larger than `max_binlog_size` if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, **mysqld** also creates a binary log index file that contains the names of all used binary log files. By default this has the same basename as the binary log file, with the extension `'.index'`. You can change the name of the binary log index file with the `--log-bin-index[=file_name]` option. You should not manually edit this file while **mysqld** is running; doing so would confuse **mysqld**.

Writes to the binary log file and binary log index file are handled in the same way as writes to MyISAM tables. See [Section A.4.3, “How MySQL Handles a Full Disk”](#).

You can delete all binary log files with the `RESET MASTER` statement, or a subset of them with `PURGE MASTER LOGS`. See [Section 13.5.5.5, “RESET Syntax”](#), and [Section 13.6.1, “SQL Statements for Controlling Master Servers”](#).

The binary log format has some known limitations that can affect recovery from backups. See [Section 6.7, “Replication Features and Known Problems”](#).

Binary logging for stored routines and triggers is done as described in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

You can use the following options to **mysqld** to affect what is logged to the binary log. See also the discussion that follows this option list.

If you are using replication, the options described here affect which statements are sent by a master server to its slaves. There are also options for slave servers that control which statements received from the master to execute or ignore. For details, see [Section 6.8, “Replication Startup Options”](#).

- `--binlog-do-db=db_name`

Tell the server to restrict binary logging to updates for which the default database is *db_name* (that is, the database selected by `USE`). All other databases that are not explicitly mentioned are ignored. If you use this option, you should ensure that you do updates only in the default database.

There is an exception to this for `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with `binlog-do-db=sales`, and you run `USE prices; UPDATE sales.january SET amount=amount+1000;`, this statement is *not* written into the binary log.

To log multiple databases, use multiple options, specifying the option once for each database.

- `--binlog-ignore-db=db_name`

Tell the server to suppress binary logging of updates for which the default database is *db_name* (that is, the database selected by `USE`). If you use this option, you should ensure that you do updates only in the default database.

As with the `--binlog-do-db` option, there is an exception for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. The server uses the database named in the statement (not the default database) to decide whether it should log the statement.

An example of what does not work as you might expect: If the server is started with `binlog-ignore-db=sales`, and you run `USE prices; UPDATE sales.january SET amount=amount+1000;`, this statement *is* written into the binary log.

To ignore multiple databases, use multiple options, specifying the option once for each database.

The server evaluates the options for logging or ignoring updates to the binary log

according to the following rules. As described previously, there is an exception for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. In those cases, the database being *created*, *altered*, or *dropped* replaces the default database in the following rules.

1. Are there `--binlog-do-db` or `--binlog-ignore-db` rules?
 - No: Write the statement to the binary log and exit.
 - Yes: Go to the next step.
2. There are some rules (`--binlog-do-db`, `--binlog-ignore-db`, or both). Is there a default database (has any database been selected by `USE`)?
 - No: Do *not* write the statement, and exit.
 - Yes: Go to the next step.
3. There is a default database. Are there some `--binlog-do-db` rules?
 - Yes: Does the default database match any of the `--binlog-do-db` rules?
 - Yes: Write the statement and exit.
 - No: Do *not* write the statement, and exit.
 - No: Go to the next step.
4. There are some `--binlog-ignore-db` rules. Does the default database match any of the `--binlog-ignore-db` rules?
 - Yes: Do not write the statement, and exit.
 - No: Write the query and exit.

For example, a slave running with only `--binlog-do-db=sales` does not write to the binary log any statement for which the default database is different from `sales` (in other words, `--binlog-do-db` can sometimes mean “ignore other databases”).

If you are using replication, you should not delete old binary log files until you are sure that no slave still needs to use them. For example, if your slaves never run more than three days behind, once a day you can execute **mysqladmin flush-logs** on the master and then remove any logs that are more than three days old. You can remove the files manually, but it is preferable to use `PURGE MASTER LOGS`, which also safely updates the binary log index file for you (and which can take a date argument). See [Section 13.6.1, “SQL Statements for Controlling Master Servers”](#).

A client that has the `SUPER` privilege can disable binary logging of its own statements by using a `SET SQL_LOG_BIN=0` statement. See [Section 13.5.3, “SET Syntax”](#).

You can display the contents of binary log files with the **mysqlbinlog** utility. This can be useful when you want to reprocess statements in the log. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log_file | mysql -h server_name
```

See [Section 8.10, “mysqlbinlog — Utility for Processing Binary Log Files”](#), for more information on the **mysqlbinlog** utility and how to use it. **mysqlbinlog** also can be used with relay log files because they are written using the same format as binary log files.

Binary logging is done immediately after a statement completes but before any locks are released or any commit is done. This ensures that the log is logged in execution order.

Updates to non-transactional tables are stored in the binary log immediately after execution. Within an uncommitted transaction, all updates (`UPDATE`, `DELETE`, or `INSERT`) that change transactional tables such as BDB or InnoDB tables are cached until a `COMMIT` statement is received by the server. At that point, **mysqld** writes the entire transaction to the binary log before the `COMMIT` is executed. When the thread that handles the transaction starts, it allocates a buffer of `binlog_cache_size` to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends.

Modifications to non-transactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to non-transactional tables, the entire

transaction is logged with a ROLLBACK statement at the end to ensure that the modifications to those tables are replicated.

The `Binlog_cache_use` status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The `Binlog_cache_disk_use` status variable shows how many of those transactions actually had to use a temporary file. These two variables can be used for tuning `binlog_cache_size` to a large enough value that avoids the use of temporary files.

The `max_binlog_cache_size` system variable (default 4GB) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this, it fails and rolls back.

If you are using the binary log, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statement. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation.

Note that the binary log format is different in MySQL 5.0 from previous versions of MySQL, due to enhancements in replication. See [Section 6.5, “Replication Compatibility Between MySQL Versions”](#).

By default, the binary log is not synchronized to disk at each write. So if the operating system or machine (not only the MySQL server) crashes, there is a chance that the last statements of the binary log are lost. To prevent this, you can make the binary log be synchronized to disk after every N writes to the binary log, with the `sync_binlog` system variable. See [Section 5.2.2, “Server System Variables”](#). 1 is the safest value for `sync_binlog`, but also the slowest. Even with `sync_binlog` set to 1, there is still the chance of an inconsistency between the table content and binary log content in case of a crash. For example, if you are using InnoDB tables and the MySQL server processes a `COMMIT` statement, it writes the whole transaction to the binary log and then commits this transaction into InnoDB. If the server crashes between those two operations, the transaction is rolled back by InnoDB at restart but still exists in the binary log. This problem can be solved with the `--innodb-safe-binlog` option, which adds consistency between the content of InnoDB tables and the binary log. (Note: `--innodb-safe-binlog` is unneeded as of MySQL 5.0; it was made obsolete by the introduction of XA transaction support.)

For this option to provide a greater degree of safety, the MySQL server should also be configured to synchronize the binary log and the InnoDB logs to disk at every transaction. The InnoDB logs are synchronized by default, and `sync_binlog=1` can be used to synchronize the binary log. The effect of this option is that at restart after a crash, after doing a rollback of transactions, the MySQL server cuts rolled back InnoDB transactions from the binary log. This ensures that the binary log reflects the exact data of InnoDB tables, and so, that the slave remains in synchrony with the master (not receiving a statement which has been rolled back).

Note that `--innodb-safe-binlog` can be used even if the MySQL server updates other storage engines than InnoDB. Only statements and transactions that affect InnoDB tables are subject to removal from the binary log at InnoDB's crash recovery. If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed InnoDB transaction. This should not happen if `sync_binlog=1` and the disk/filesystem do an actual sync when they are requested to (some don't), so the server prints an error message `The binary log <name> is shorter than its expected size`. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the master's data.

5.12.4. The Slow Query Log

The slow query log consists of all SQL statements that took more than `long_query_time` seconds to execute. The time to acquire the initial table locks is not counted as execution time. The minimum and default values of `long_query_time` are 1 and 10, respectively.

mysqld writes a statement to the slow query log after it has been executed and after all locks have been released. Log order may be different from execution order.

To enable the slow query log, start **mysqld** with the `--log-slow-queries[=file_name]` option.

If no *file_name* value is given, the default is the name of the host machine with a suffix of `-slow.log`. If a filename is given, but not as an absolute pathname, the server writes the file in the data directory.

The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can become a difficult task. To make this easier, you can process the slow query log using the **mysqldumpslow** command to summarize the queries that appear in the log. Use **mysqldumpslow --help** to see the options that this command supports.

In MySQL 5.0, queries that do not use indexes are logged in the slow query log if the `--log-queries-not-using-indexes` option is specified. See [Section 5.2.1, “mysqld Command Options”](#).

In MySQL 5.0, the `--log-slow-admin-statements` server option enables you to request logging of slow administrative statements such as `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `ALTER TABLE` to the slow query log.

Queries handled by the query cache are not added to the slow query log, nor are queries that would not benefit from the presence of an index because the table has zero rows or one row.

5.12.5. Server Log Maintenance

MySQL Server can create a number of different log files that make it easy to see what is going on. See [Section 5.12, “MySQL Server Logs”](#). However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See [Section 5.10.1, “Database Backups”](#).

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for this. If you installed MySQL from an RPM distribution, this script should have been installed automatically. You should be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all slaves.

On other systems, you must install a short script yourself that you start from **cron** (or its equivalent) for handling log files.

You can force MySQL to start using new log files by using **mysqladmin flush-logs** or by using the SQL statement `FLUSH LOGS`.

A log flushing operation does the following:

- If general query logging (`--log`) or slow query logging (`--log-slow-queries`) is used, the server closes and reopens the general query log file or slow query log file.
- If binary logging (`--log-bin`) is used, the server closes the current log file and opens a new log file with the next sequence number.

The server creates a new binary log file when you flush the logs. However, it just closes and reopens the general and slow query log files. To cause new files to be created on Unix, rename the current logs before flushing them. At flush time, the server will open new logs with the original names. For example, if the general and slow query logs are named `mysql.log` and `mysql-slow.log`, you can use a series of commands like this:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mv mysql-slow.log mysql-slow.old
shell> mysqladmin flush-logs
```

At this point, you can make a backup of `mysql.old` and `mysql-slow.log` and then remove them from disk.

On Windows, you cannot rename log files while the server has them open. You must stop the server and rename them, and then restart the server to create new logs.

5.13. Running Multiple MySQL Servers on the Same Machine

In some cases, you might want to run multiple **mysqld** servers on the same machine. You might want to test a new MySQL release while leaving your existing production setup undisturbed. Or you might want to give different users access to different **mysqld** servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

To run multiple servers on a single machine, each server must have unique values for several operating parameters. These can be set on the command line or in option files. See [Section 4.3, “Specifying Program Options”](#).

At least the following options must be different for each server:

- `--port=port_num`

`--port` controls the port number for TCP/IP connections.

- `--socket=path`

`--socket` controls the Unix socket file path on Unix and the name of the named pipe on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers that support named-pipe connections.

- `--shared-memory-base-name=name`

This option currently is used only on Windows. It designates the shared-memory name used by a Windows server to allow clients to connect via shared memory. It is necessary to specify distinct shared-memory names only for those servers that support shared-memory connections.

- `--pid-file=file_name`

This option is used only on Unix. It indicates the pathname of the file in which the server writes its process ID.

If you use the following log file options, they must be different for each server:

- `--log=file_name`
- `--log-bin=file_name`
- `--log-update=file_name`
- `--log-error=file_name`
- `--bdb-logdir=file_name`

[Section 5.12.5, “Server Log Maintenance”](#), discusses the log file options further.

For better performance, you can specify the following options differently for each server, to spread the load between several physical disks:

- `--tmpdir=path`
- `--bdb-tmpdir=path`

Having different temporary directories is also recommended to make it easier to determine which MySQL server created any given temporary file.

With very limited exceptions, each server should use a different data directory, which is specified using the `--datadir=path` option.

Warning: Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log filenames that are unique to each server. Otherwise, the servers try to log to the same files. Please note that this kind of setup only works with MyISAM and MERGE tables, and not with any of the other storage engines.

The warning against sharing a data directory among servers also applies in an NFS environment. Allowing multiple MySQL servers to access a common data directory over NFS is a *very bad idea*.

- The primary problem is that NFS is the speed bottleneck. It is not meant for such use.
- Another risk with NFS is that you must devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the `lockd` daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Make it easy for yourself: Forget about sharing a data directory among servers over NFS. A better solution is to have one computer that contains several CPUs and use an operating system that handles threads efficiently.

If you have multiple MySQL installations in different locations, you can specify the base installation directory for each server with the `--basedir=path` option to cause each server to use a different data directory, log files, and PID file. (The defaults for all these values are determined relative to the base directory). In that case, the only other options you need to specify are the `--socket` and `--port` options. For example, suppose that you install different versions of MySQL using tar file binary distributions. These install in different locations, so you can start the server for each installation using the command **`bin/mysqld_safe`** under its corresponding base directory. **`mysqld_safe`** determines the proper `--basedir` option to pass to **`mysqld`**, and you need specify only the `--socket` and `--port` options to **`mysqld_safe`**.

As discussed in the following sections, it is possible to start additional servers by setting environment variables or by specifying appropriate command-line options. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The `--defaults-file` option is useful for this purpose.

5.13.1. Running Multiple Servers on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters. On Windows NT-based systems, you also have the option of installing several servers as Windows services and running them that way. General instructions for running MySQL servers from the command line or as services are given in [Section 2.3, “Installing MySQL on Windows”](#). This section describes how to make sure that

you start each server with different values for those startup options that must be unique per server, such as the data directory. These options are described in [Section 5.13, “Running Multiple MySQL Servers on the Same Machine”](#).

5.13.1.1. Starting Multiple Windows Servers at the Command Line

To start multiple servers manually from the command line, you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the filename with a `--defaults-file` option when you run it.

Suppose that you want to run **mysqld** on port 3307 with a data directory of `C:\mydata1`, and **mysqld-max** on port 3308 with a data directory of `C:\mydata2`. (To do this, make sure that before you start the servers, each data directory exists and has its own copy of the `mysql` database that contains the grant tables.) Then create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

Create a second file named `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

Then start each server with its own option file:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-max --defaults-file=C:\my-opts2.cnf
```

On NT, each server starts in the foreground (no new prompt appears until the server exits later), so you will need to issue those two commands in separate console windows.

To shut down the servers, you must connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 shutdown
```

Servers configured as just described allow clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to allow named-pipe connections, use the **mysqld-nt** or **mysqld-max-nt** servers and specify options that enable the named pipe and specify its name. Each server that supports named-pipe connections must use a unique pipe name. For example, the C:\my-opts1.cnf file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

Then start the server this way:

```
C:\> C:\mysql\bin\mysqld-nt --defaults-file=C:\my-opts1.cnf
```

Modify C:\my-opts2.cnf similarly for use by the second server.

A similar procedure applies for servers that you want to support shared-memory connections. Enable such connections with the `--shared-memory` option and specify a unique shared-memory name for each server with the `--shared-memory-base-name` option.

5.13.1.2. Starting Multiple Windows Servers as Services

On NT-based systems, a MySQL server can run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in [Section 2.3.11, “Starting MySQL as a Windows Service”](#).

You can also install multiple MySQL servers as services. In this case, you must make sure that each server uses a different service name in addition to all the other parameters that must be unique for each server.

For the following instructions, assume that you want to run the **mysqld-nt** server from two different versions of MySQL that are installed at C:\mysql-4.1.8 and C:\mysql-5.0.25, respectively. (This might be the case if you're running 4.1.8 as your production server, but also want to conduct tests using 5.0.25.)

The following principles apply when installing a MySQL service with the `--install` or `--install-manual` option:

- If you specify no service name, the server uses the default service name of `mysql` and the server reads options from the `[mysqld]` group in the standard option files.
- If you specify a service name after the `--install` option, the server ignores the `[mysqld]` option group and instead reads options from the group that has the same name as the service. The server reads options from the standard option files.
- If you specify a `--defaults-file` option after the service name, the server ignores the standard option files and reads options only from the `[mysqld]` group of the named file.

Note: Before MySQL 4.0.17, only a server installed using the default service name (`MySQL`) or one installed explicitly with a service name of `mysqld` read the `[mysqld]` group in the standard option files. As of 4.0.17, all servers read the `[mysqld]` group if they read the standard option files, even if they are installed using another service name. This allows you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group named after each service for use by the server installed with that service name.

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, be sure that you shut down and remove any existing MySQL services first.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 4.1.8 `mysqld-nt` using the service name of `mysqld1` and the 5.0.25 `mysqld-nt` using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 4.1.8 and the `[mysqld2]` group for 5.0.25. For example, you can set up `C:\my.cnf` like this:

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-4.1.8
port = 3307
enable-named-pipe
```

```
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-5.0.25
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows, using the full server pathnames to ensure that Windows registers the correct executable program for each service:

```
C:\> C:\mysql-4.1.8\bin\mysqld-nt --install mysqld1
C:\> C:\mysql-5.0.25\bin\mysqld-nt --install mysqld2
```

To start the services, use the services manager, or use **NET START** with the appropriate service names:

```
C:\> NET START mysqld1
C:\> NET START mysqld2
```

To stop the services, use the services manager, or use **NET STOP** with the appropriate service names:

```
C:\> NET STOP mysqld1
C:\> NET STOP mysqld2
```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

With this approach, to specify options for the 4.1.8 **mysqld-nt**, create a file `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
basedir = C:/mysql-4.1.8
port = 3307
enable-named-pipe
socket = mypipe1
```

For the 5.0.25 **mysqld-nt**, create a file `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
basedir = C:/mysql-5.0.25
```

```
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows (enter each command on a single line):

```
C:\> C:\mysql-4.1.8\bin\mysqld-nt --install mysqld1
      --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-5.0.25\bin\mysqld-nt --install mysqld2
      --defaults-file=C:\my-opts2.cnf
```

To use a `--defaults-file` option when you install a MySQL server as a service, you must precede the option with the service name.

After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use `mysqld --remove` for each one, specifying a service name following the `--remove` option. If the service name is the default (MySQL), you can omit it.

5.13.2. Running Multiple Servers on Unix

The easiest way to run multiple servers on Unix is to compile them with different TCP/IP ports and Unix socket files so that each one is listening on different network interfaces. Compiling in different base directories for each installation also results automatically in a separate, compiled-in data directory, log file, and PID file location for each server.

Assume that an existing 4.1.8 server is configured for the default TCP/IP port number (3306) and Unix socket file (`/tmp/mysql.sock`). To configure a new 5.0.25 server to have different operating parameters, use a **configure** command something like this:

```
shell> ./configure --with-tcp-port=port_number \
              --with-unix-socket-path=file_name \
              --prefix=/usr/local/mysql-5.0.25
```

Here, *port_number* and *file_name* must be different from the default TCP/IP port number and Unix socket file pathname, and the `--prefix` value should specify an installation directory different from the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket filename:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

Note that if you specify localhost as a hostname, **mysqladmin** defaults to using a Unix socket file connection rather than TCP/IP. From MySQL 4.1 onward, you can explicitly specify the connection protocol to use by using the `--protocol={TCP|SOCKET|PIPE|MEMORY}` option.

You don't have to compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to use the same server binary and start each invocation of it with different parameter values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` and `--port` option values, and pass a `--datadir=path` option to **mysqld_safe** so that the server uses a different data directory.

Another way to achieve a similar effect is to use environment variables to set the Unix socket filename and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server.

[Appendix F, *Environment Variables*](#), includes a list of other environment

variables you can use to affect **mysqld**.

For automatic server execution, the startup script that is executed at boot time should execute the following command once for each server with an appropriate option file path for each command:

```
shell> mysqld_safe --defaults-file=file_name
```

Each option file should contain option values specific to a given server.

On Unix, the **mysqld_multi** script is another way to start multiple servers. See [Section 5.4.3, “mysqld_multi — Manage Multiple MySQL Servers”](#).

5.13.3. Using Client Programs in a Multiple-Server Environment

To connect with a client program to a MySQL server that is listening to different network interfaces from those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name --port=port_number` to connect via TCP/IP to a remote server, with `--host=127.0.0.1 --port=port_number` to connect via TCP/IP to a local server, or with `--host=localhost --socket=file_name` to connect to a local server via a Unix socket file or a Windows named pipe.
- As of MySQL 4.1, start the client with `--protocol=tcp` to connect via TCP/IP, `--protocol=socket` to connect via a Unix socket file, `--protocol=pipe` to connect via a named pipe, or `--protocol=memory` to connect via shared memory. For TCP/IP connections, you may also need to specify `--host` and `--port` options. For the other types of connections, you may need to specify a `--socket` option to specify a Unix socket file or Windows named-pipe name, or a `--shared-memory-base-name` option to specify the shared-memory name. Shared-memory connections are supported only on Windows.
- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See [Appendix F](#),

[Environment Variables.](#)

- Specify the default Unix socket file and TCP/IP port number in the [client] group of an option file. For example, you can use C:\my.cnf on Windows, or the .my.cnf file in your home directory on Unix. See [Section 4.3.2, “Using Option Files”](#).
- In a C program, you can specify the socket file or port number arguments in the mysql_real_connect() call. You can also have the program read option files by calling mysql_options(). See [Section 22.2.3, “C API Function Descriptions”](#).
- If you are using the Perl DBD::mysql module, you can read options from MySQL option files. For example:

```
$dsn = "DBI:mysql:test;mysql_read_default_group=client;"  
      . "mysql_read_default_file=/usr/local/mysql/data/my.cnf"  
$dbh = DBI->connect($dsn, $user, $password);
```

See [Section 22.4, “MySQL Perl API”](#).

Other programming interfaces may provide similar capabilities for reading option files.

5.14. The MySQL Query Cache

The query cache stores the text of a `SELECT` statement together with the corresponding result that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again.

The query cache is extremely useful in an environment where you have tables that do not change very often and for which the server receives many identical queries. This is a typical situation for many Web servers that generate many dynamic pages based on database content.

Note: The query cache does not return stale data. When tables are modified, any relevant entries in the query cache are flushed.

Note: The query cache does not work in an environment where you have multiple **mysqld** servers updating the same MyISAM tables.

Note: The query cache is not used for server-side prepared statements. If you're using server-side prepared statements consider that these statement won't be satisfied by the query cache. See [Section 22.2.4, “C API Prepared Statements”](#).

Some performance data for the query cache follows. These results were generated by running the MySQL benchmark suite on a Linux Alpha 2×500MHz system with 2GB RAM and a 64MB query cache.

- If all the queries you are performing are simple (such as selecting a row from a table with one row), but still differ so that the queries cannot be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. In real life, queries tend to be much more complicated, so the overhead normally is significantly lower.
- Searches for a single row in a single-row table are 238% faster with the query cache than without it. This can be regarded as close to the minimum speedup to be expected for a query that is cached.

To disable the query cache at server startup, set the `query_cache_size` system variable to 0. By disabling the query cache code, there is no noticeable overhead.

If you build MySQL from source, query cache capabilities can be excluded from the server entirely by invoking **configure** with the `--without-query-cache` option.

5.14.1. How the Query Cache Operates

This section describes how the query cache works when it is operational. [Section 5.14.3, “Query Cache Configuration”](#), describes how to control whether it is operational.

Incoming queries are compared to those in the query cache before parsing, so the following two queries are regarded as different by the query cache:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Queries must be *exactly* the same (byte for byte) to be seen as identical. In addition, query strings that are identical may be treated as different for other reasons. Queries that use different databases, different protocol versions, or different default character sets are considered different queries and are cached separately.

Before a query result is fetched from the query cache, MySQL checks that the user has `SELECT` privilege for all databases and tables involved. If this is not the case, the cached result is not used.

If a query result is returned from query cache, the server increments the `Qcache_hits` status variable, not `Com_select`. See [Section 5.14.4, “Query Cache Status and Maintenance”](#).

If a table changes, all cached queries that use the table become invalid and are removed from the cache. This includes queries that use `MERGE` tables that map to the changed table. A table can be changed by many types of statements, such as `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `ALTER TABLE`, `DROP TABLE`, or `DROP DATABASE`.

Transactional InnoDB tables that have been changed are invalidated when a `COMMIT` is performed.

The query cache also works within transactions when using InnoDB tables,

making use of the table version number to detect whether its contents are still current.

In MySQL 5.0, queries generated by views are cached.

Before MySQL 5.0, a query that began with a leading comment could be cached, but could not be fetched from the cache. This problem is fixed in MySQL 5.0.

The query cache works for `SELECT SQL_CALC_FOUND_ROWS ...` and `SELECT FOUND_ROWS()` type queries. `FOUND_ROWS()` returns the correct value even if the preceding query was fetched from the cache because the number of found rows is also stored in the cache.

A query cannot be cached if it contains any of the functions shown in the following table.

<code>BENCHMARK()</code>	<code>CONNECTION_ID()</code>	<code>CURDATE()</code>
<code>CURRENT_DATE()</code>	<code>CURRENT_TIME()</code>	<code>CURRENT_TIMESTAMP()</code>
<code>CURTIME()</code>	<code>DATABASE()</code>	<code>ENCRYPT()</code> with one parameter
<code>FOUND_ROWS()</code>	<code>GET_LOCK()</code>	<code>LAST_INSERT_ID()</code>
<code>LOAD_FILE()</code>	<code>MASTER_POS_WAIT()</code>	<code>NOW()</code>
<code>RAND()</code>	<code>RELEASE_LOCK()</code>	<code>SYSDATE()</code>
<code>UNIX_TIMESTAMP()</code> with no parameters	<code>USER()</code>	

A query also is not cached under these conditions:

- It refers to user-defined functions (UDFs).
- It refers to user variables.
- It refers to tables in the `mysql` system database.
- It is of any of the following forms:

```
SELECT ... IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
```

```
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value. See the MyODBC section of [Chapter 23, Connectors](#).

- It was issued as a prepared statement, even if no placeholders were employed. For example, the query used here is not cached:

```
char *my_sql_stmt = "SELECT a, b FROM table_c";  
/* ... */  
mysql_stmt_prepare(stmt, my_sql_stmt, strlen(my_sql_stmt));
```

See [Section 22.2.4, “C API Prepared Statements”](#).

- It uses TEMPORARY tables.
- It does not use any tables.
- The user has a column-level privilege for any of the involved tables.

5.14.2. Query Cache SELECT Options

Two query cache-related options may be specified in SELECT statements:

- SQL_CACHE

The query result is cached if the value of the `query_cache_type` system variable is ON or DEMAND.

- SQL_NO_CACHE

The query result is not cached.

Examples:

```
SELECT SQL_CACHE id, name FROM customer;  
SELECT SQL_NO_CACHE id, name FROM customer;
```

5.14.3. Query Cache Configuration

The `have_query_cache` server system variable indicates whether the query cache is available:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES |
+-----+-----+
```

When using a standard MySQL binary, this value is always YES, even if query caching is disabled.

Several other system variables control query cache operation. These can be set in an option file or on the command line when starting **mysqld**. The query cache system variables all have names that begin with `query_cache_`. They are described briefly in [Section 5.2.2, “Server System Variables”](#), with additional configuration information given here.

To set the size of the query cache, set the `query_cache_size` system variable. Setting it to 0 disables the query cache. The default size is 0, so the query cache is disabled by default.

When you set `query_cache_size` to a non-zero value, keep in mind that the query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value too small, you'll get a warning, as in this example:

```
mysql> SET GLOBAL query_cache_size = 40000;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1282
Message: Query cache failed to set size 39936; new query cache size
```

```
mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SHOW VARIABLES LIKE 'query_cache_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
```

```
| query_cache_size | 41984 |  
+-----+-----+
```

If the query cache size is greater than 0, the `query_cache_type` variable influences how it works. This variable can be set to the following values:

- A value of 0 or `OFF` prevents caching or retrieval of cached results.
- A value of 1 or `ON` allows caching except of those statements that begin with `SELECT SQL_NO_CACHE`.
- A value of 2 or `DEMAND` causes caching of only those statements that begin with `SELECT SQL_CACHE`.

Setting the `GLOBAL query_cache_type` value determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the `SESSION query_cache_type` value. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

To control the maximum size of individual query results that can be cached, set the `query_cache_limit` system variable. The default value is 1MB.

When a query that is to be cached, its result (the data sent to the client) is stored in the query cache during result retrieval. Therefore the data usually is not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (timewise), the query cache allocates blocks with a minimum size given by the `query_cache_min_res_unit` system variable. When a query is executed, the last result block is trimmed to the actual data size so that unused memory is freed. Depending on the types of queries your server executes, you might find it helpful to tune the value of `query_cache_min_res_unit`:

- The default value of `query_cache_min_res_unit` is 4KB. This should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation, as indicated by a large number of free

blocks. Fragmentation can force the query cache to prune (delete) queries from the cache due to lack of memory. In this case, you should decrease the value of `query_cache_min_res_unit`. The number of free blocks and queries removed due to pruning are given by the values of the `Qcache_free_blocks` and `Qcache_lowmem_prunes` status variables.

- If most of your queries have large results (check the `Qcache_total_blocks` and `Qcache_queries_in_cache` status variables), you can increase performance by increasing `query_cache_min_res_unit`. However, be careful to not make it too large (see the previous item).

5.14.4. Query Cache Status and Maintenance

You can check whether the query cache is present in your MySQL server using the following statement:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES |
+-----+-----+
```

You can defragment the query cache to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The `RESET QUERY CACHE` statement removes all query results from the query cache. The `FLUSH TABLES` statement also does this.

To monitor query cache performance, use `SHOW STATUS` to view the cache status variables:

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Qcache_free_blocks     | 36    |
| Qcache_free_memory     | 138488 |
| Qcache_hits            | 79570 |
| Qcache_inserts         | 27087 |
| Qcache_lowmem_prunes   | 3114  |
| Qcache_not_cached      | 22989 |
+-----+-----+
```

```

| Qcache_queries_in_cache | 415 |
| Qcache_total_blocks     | 912 |
+-----+-----+

```

Descriptions of each of these variables are given in [Section 5.2.4, “Server Status Variables”](#). Some uses for them are described here.

The total number of SELECT queries is given by this formula:

```

Com_select
+ Qcache_hits
+ queries with errors found by parser

```

The Com_select value is given by this formula:

```

Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during the column-privileges check

```

The query cache uses variable-length blocks, so Qcache_total_blocks and Qcache_free_blocks may indicate query cache memory fragmentation. After FLUSH QUERY CACHE, only a single free block remains.

Every cached query requires a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query requires one block. However, if two or more queries use the same table, only one table block needs to be allocated.

The information provided by the Qcache_lowmem_prunes status variable can help you tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a least recently used (LRU) strategy to decide which queries to remove from the cache. Tuning information is given in [Section 5.14.3, “Query Cache Configuration”](#).

Chapter 6. Replication

Table of Contents

- [6.1. Introduction to Replication](#)
- [6.2. Replication Implementation Overview](#)
- [6.3. Replication Implementation Details](#)
 - [6.3.1. Replication Master Thread States](#)
 - [6.3.2. Replication Slave I/O Thread States](#)
 - [6.3.3. Replication Slave SQL Thread States](#)
 - [6.3.4. Replication Relay and Status Files](#)
- [6.4. How to Set Up Replication](#)
- [6.5. Replication Compatibility Between MySQL Versions](#)
- [6.6. Upgrading a Replication Setup](#)
 - [6.6.1. Upgrading Replication to 5.0](#)
- [6.7. Replication Features and Known Problems](#)
- [6.8. Replication Startup Options](#)
- [6.9. How Servers Evaluate Replication Rules](#)
- [6.10. Replication FAQ](#)
- [6.11. Troubleshooting Replication](#)
- [6.12. How to Report Replication Bugs or Problems](#)
- [6.13. Auto-Increment in Multiple-Master Replication](#)

This chapter describes the various replication features provided by MySQL. It introduces replication concepts, shows how to set up replication servers, and serves as a reference to the available replication options. It also provides a list of frequently asked questions (with answers), and troubleshooting advice for solving replication problems.

For a description of the syntax of replication-related SQL statements, see [Section 13.6, “Replication Statements”](#).

6.1. Introduction to Replication

MySQL features support for one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves. This is in contrast to the *synchronous* replication which is a characteristic of MySQL Cluster (see [Chapter 15, MySQL Cluster](#)).

In single-master replication, the master server writes updates to its binary log files and maintains an index of those files to keep track of log rotation. The binary log files serve as a record of updates to be sent to any slave servers. When a slave connects to its master, it informs the master of the position up to which the slave read the logs at its last successful update. The slave receives any updates that have taken place since that time, and then blocks and waits for the master to notify it of new updates.

A slave server can itself serve as a master if you want to set up chained replication servers.

Multiple-master replication is possible, but raises issues not present in single-master replication. See [Section 6.13, “Auto-Increment in Multiple-Master Replication”](#).

When you are using replication, all updates to the tables that are replicated should be performed on the master server. Otherwise, you must always be careful to avoid conflicts between updates that users make to tables on the master and updates that they make to tables on the slave.

Replication offers benefits for robustness, speed, and system administration:

- Robustness is increased with a master/slave setup. In the event of problems with the master, you can switch to the slave as a backup.
- Better response time for clients can be achieved by splitting the load for processing client queries between the master and slave servers. `SELECT` queries may be sent to the slave to reduce the query processing load of the master. Statements that modify data should still be sent to the master so that the master and slave do not get out of synchrony. This load-balancing strategy is effective if non-updating queries dominate, but that is the normal

case.

- Another benefit of using replication is that you can perform database backups using a slave server without disturbing the master. The master continues to process updates while the backup is being made. See [Section 5.10.1, “Database Backups”](#).

6.2. Replication Implementation Overview

MySQL replication is based on the master server keeping track of all changes to your databases (updates, deletes, and so on) in its binary logs. Therefore, to use replication, you must enable binary logging on the master server. See [Section 5.12.3, “The Binary Log”](#).

Each slave server receives from the master the saved updates that the master has recorded in its binary log, so that the slave can execute the same updates on its copy of the data.

It is *extremely* important to realize that the binary log is simply a record starting from the fixed point in time at which you enable binary logging. Any slaves that you set up need copies of the databases on your master *as they existed at the moment you enabled binary logging on the master*. If you start your slaves with databases that are not in the same state as those on the master when the binary log was started, your slaves are quite likely to fail.

One way to copy the master's data to the slave is to use the `LOAD DATA FROM MASTER` statement. However, `LOAD DATA FROM MASTER` works only if all the tables on the master use the `MyISAM` storage engine. In addition, this statement acquires a global read lock, so no updates on the master are possible while the tables are being transferred to the slave. When we implement lock-free hot table backup, this global read lock will no longer be necessary.

Due to these limitations, we recommend that at this point you use `LOAD DATA FROM MASTER` only if the dataset on the master is relatively small, or if a prolonged read lock on the master is acceptable. Although the actual speed of `LOAD DATA FROM MASTER` may vary from system to system, a good rule of thumb for how long it takes is 1 second per 1MB of data. This is a rough estimate, but you should find it fairly accurate if both master and slave are equivalent to 700MHz Pentium CPUs in performance and are connected through a 100Mbps network.

After the slave has been set up with a copy of the master's data, it connects to the master and waits for updates to process. If the master fails, or the slave loses connectivity with your master, the slave keeps trying to connect periodically until it is able to resume listening for updates. The `--master-connect-retry`

option controls the retry interval. The default is 60 seconds.

Each slave keeps track of where it left off when it last read from its master server. The master has no knowledge of how many slaves it has or which ones are up to date at any given time.

6.3. Replication Implementation Details

MySQL replication capabilities are implemented using three threads (one on the master server and two on the slave). When a `START SLAVE` statement is issued on a slave server, the slave creates an I/O thread, which connects to the master and asks it to send the updates recorded in its binary logs. The master creates a thread to send the binary log contents to the slave. This thread can be identified as the `Binlog Dump` thread in the output of `SHOW PROCESSLIST` on the master. The slave I/O thread reads the updates that the master `Binlog Dump` thread sends and copies them to local files, known as *relay logs*, in the slave's data directory. The third thread is the SQL thread, which the slave creates to read the relay logs and to execute the updates they contain.

In the preceding description, there are three threads per master/slave connection. A master that has multiple slaves creates one thread for each currently-connected slave, and each slave has its own I/O and SQL threads.

The slave uses two threads so that reading updates from the master and executing them can be separated into two independent tasks. Thus, the task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is stored locally in the slave's relay logs, ready for execution the next time that the slave starts. This enables the master server to purge its binary logs sooner because it no longer needs to wait for the slave to fetch their contents.

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the master and on the slave regarding replication. The following example illustrates how the three threads show up in the output from `SHOW PROCESSLIST`.

On the master server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*****
1. row *****
```

```
    Id: 2
    User: root
    Host: localhost:32931
    db: NULL
Command: Binlog Dump
    Time: 94
    State: Has sent all binlog to slave; waiting for binlog to
           be updated
    Info: NULL
```

Here, thread 2 is a Binlog Dump replication thread for a connected slave. The state information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur. If you see no Binlog Dump threads on a master server, this means that replication is not running — that is, that no slaves are currently connected.

On the slave server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
    Id: 10
    User: system user
    Host:
    db: NULL
Command: Connect
    Time: 11
    State: Waiting for master to send event
    Info: NULL
***** 2. row *****
    Id: 11
    User: system user
    Host:
    db: NULL
Command: Connect
    Time: 11
    State: Has read all relay log; waiting for the slave I/O
           thread to update it
    Info: NULL
```

This information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. At the time that the `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the slave is compared to the master. See [Section 6.10, “Replication FAQ”](#).

6.3.1. Replication Master Thread States

The following list shows the most common states you may see in the State column for the master's Binlog Dump thread. If you see no Binlog Dump threads on a master server, this means that replication is not running — that is, that no slaves are currently connected.

- Sending binlog event to slave

Binary logs consist of *events*, where an event is usually an update plus some other information. The thread has read an event from the binary log and is now sending it to the slave.

- Finished reading one binlog; switching to next binlog

The thread has finished reading a binary log file and is opening the next one to send to the slave.

- Has sent all binlog to slave; waiting for binlog to be updated

The thread has read all outstanding updates from the binary logs and sent them to the slave. The thread is now idle, waiting for new events to appear in the binary log resulting from new updates occurring on the master.

- Waiting to finalize termination

A very brief state that occurs as the thread is stopping.

6.3.2. Replication Slave I/O Thread States

The following list shows the most common states you see in the State column for a slave server I/O thread. This state also appears in the Slave_IO_State column displayed by SHOW SLAVE STATUS, so you can get a good view of what is happening by using that statement.

- Connecting to master

The thread is attempting to connect to the master.

- Checking master version

A state that occurs very briefly, after the connection to the master is established.

- Registering slave on master

A state that occurs very briefly after the connection to the master is established.

- Requesting binlog dump

A state that occurs very briefly, after the connection to the master is established. The thread sends to the master a request for the contents of its binary logs, starting from the requested binary log filename and position.

- Waiting to reconnect after a failed binlog dump request

If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `--master-connect-retry` option.

- Reconnecting after a failed binlog dump request

The thread is trying to reconnect to the master.

- Waiting for master to send event

The thread has connected to the master and is waiting for binary log events to arrive. This can last for a long time if the master is idle. If the wait lasts for `slave_read_timeout` seconds, a timeout occurs. At that point, the thread considers the connection to be broken and makes an attempt to reconnect.

- Queueing master event to the relay log

The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- Waiting to reconnect after a failed master event read

An error occurred while reading (due to disconnection). The thread is sleeping for `master-connect-retry` seconds before attempting to reconnect.

- Reconnecting after a failed master event read

The thread is trying to reconnect to the master. When connection is established again, the state becomes waiting for master to send event.

- Waiting for the slave SQL thread to free enough relay log space

You are using a non-zero `relay_log_space_limit` value, and the relay logs have grown large enough that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- Waiting for slave mutex on exit

A state that occurs briefly as the thread is stopping.

6.3.3. Replication Slave SQL Thread States

The following list shows the most common states you may see in the State column for a slave server SQL thread:

- Reading event from the relay log

The thread has read an event from the relay log so that the event can be processed.

- Has read all relay log; waiting for the slave I/O thread to update it

The thread has processed all events in the relay log files, and is now waiting for the I/O thread to write new events to the relay log.

- Waiting for slave mutex on exit

A very brief state that occurs as the thread is stopping.

The State column for the I/O thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and is executing it.

6.3.4. Replication Relay and Status Files

By default, relay logs filenames have the form `host_name-relay-bin.nnnnnn`, where `host_name` is the name of the slave server host and `nnnnnn` is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `000001`. The slave uses an index file to track the relay log files currently in use. The default relay log index filename is `host_name-relay-bin.index`. By default, the slave server creates relay log files in its data directory. The default filenames can be overridden with the `--relay-log` and `--relay-log-index` server options. See [Section 6.8, “Replication Startup Options”](#).

Relay logs have the same format as binary logs and can be read using **mysqlbinlog**. The SQL thread automatically deletes each relay log file as soon as it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread takes care of doing so. However, `FLUSH LOGS` rotates relay logs, which influences when the SQL thread deletes them.

A slave server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed; for example, with `FLUSH LOGS` or **mysqladmin flush-logs**.
- When the size of the current relay log file becomes too large. The meaning of “too large” is determined as follows:
 - If the value of `max_relay_log_size` is greater than 0, that is the maximum relay log file size.
 - If the value of `max_relay_log_size` is 0, `max_binlog_size` determines the maximum relay log file size.

A slave replication server creates two additional small files in the data directory.

These *status files* are named `master.info` and `relay-log.info` by default. Their names can be changed by using the `--master-info-file` and `--relay-log-info-file` options. See [Section 6.8, “Replication Startup Options”](#).

The two status files contain information like that shown in the output of the `SHOW SLAVE STATUS` statement, which is discussed in [Section 13.6.2, “SQL Statements for Controlling Slave Servers”](#). Because the status files are stored on disk, they survive a slave server's shutdown. The next time the slave starts up, it reads the two files to determine how far it has proceeded in reading binary logs from the master and in processing its own relay logs.

The I/O thread updates the `master.info` file. The following table shows the correspondence between the lines in the file and the columns displayed by `SHOW SLAVE STATUS`.

Line	Description
1	Number of lines in the file
2	Master_Log_File
3	Read_Master_Log_Pos
4	Master_Host
5	Master_User
6	Password (not shown by <code>SHOW SLAVE STATUS</code>)
7	Master_Port
8	Connect_Retry
9	Master_SSL_Allowed
10	Master_SSL_CA_File
11	Master_SSL_CA_Path
12	Master_SSL_Cert
13	Master_SSL_Cipher
14	Master_SSL_Key

The SQL thread updates the `relay-log.info` file. The following table shows the correspondence between the lines in the file and the columns displayed by `SHOW SLAVE STATUS`.

--	--

Line	Description
1	Relay_Log_File
2	Relay_Log_Pos
3	Relay_Master_Log_File
4	Exec_Master_Log_Pos

When you back up the slave's data, you should back up these two status files as well, along with the relay log files. They are needed to resume replication after you restore the slave's data. If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. Of course, this requires that the binary logs still exist on the master server.

If your slave is subject to replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations. The directory location is specified using the `--slave-load-tmpdir` option. If this option is not specified, the directory location is the value of the `tmpdir` system variable.

6.4. How to Set Up Replication

This section briefly describes how to set up complete replication of a MySQL server. It assumes that you want to replicate all databases on the master and have not previously configured replication. You must shut down your master server briefly to complete the steps outlined here.

This procedure is written in terms of setting up a single slave, but you can repeat it to set up multiple slaves.

Although this method is the most straightforward way to set up a slave, it is not the only one. For example, if you have a snapshot of the master's data, and the master already has its server ID set and binary logging enabled, you can set up a slave without shutting down the master or even blocking updates to it. For more details, please see [Section 6.10, “Replication FAQ”](#).

If you want to administer a MySQL replication setup, we suggest that you read this entire chapter through and try all statements mentioned in [Section 13.6.1, “SQL Statements for Controlling Master Servers”](#), and [Section 13.6.2, “SQL Statements for Controlling Slave Servers”](#). You should also familiarize yourself with the replication startup options described in [Section 6.8, “Replication Startup Options”](#).

Note: This procedure and some of the replication SQL statements shown in later sections require the SUPER privilege.

1. Make sure that the versions of MySQL installed on the master and slave are compatible according to the table shown in [Section 6.5, “Replication Compatibility Between MySQL Versions”](#). Ideally, you should use the most recent version of MySQL on both master and slave.

If you encounter a problem, please do not report it as a bug until you have verified that the problem is present in the latest MySQL release.

2. Set up an account on the master server that the slave server can use to connect. This account must be given the REPLICATION SLAVE privilege. If the account is used only for replication (which is recommended), you don't need to grant any additional privileges.

Suppose that your domain is `mydomain.com` and that you want to create an account with a username of `rep1` such that slave servers can use the account to access the master server from any host in your domain using a password of `slavepass`. To create the account, use this `GRANT` statement:

```
mysql> GRANT REPLICATION SLAVE ON *.*  
      -> TO 'rep1'@'%mydomain.com' IDENTIFIED BY 'slavepass';
```

If you plan to use the `LOAD TABLE FROM MASTER` or `LOAD DATA FROM MASTER` statements from the slave host, you must grant this account additional privileges:

- Grant the account the `SUPER` and `RELOAD` global privileges.
- Grant the `SELECT` privilege for all tables that you want to load. Any master tables from which the account cannot `SELECT` will be ignored by `LOAD DATA FROM MASTER`.

For additional information about setting up user accounts and privileges, see [Section 5.9, “MySQL User Account Management”](#).

3. Flush all the tables and block write statements by executing a `FLUSH TABLES WITH READ LOCK` statement:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

For InnoDB tables, note that `FLUSH TABLES WITH READ LOCK` also blocks `COMMIT` operations. When you have acquired a global read lock, you can start a filesystem snapshot of your InnoDB tables. Internally (inside the InnoDB storage engine) the snapshot won't be consistent (because the InnoDB caches are not flushed), but this is not a cause for concern, because InnoDB resolves this at startup and delivers a consistent result. This means that InnoDB can perform crash recovery when started on this snapshot, without corruption. However, there is no way to stop the MySQL server while insuring a consistent snapshot of your InnoDB tables.

Leave running the client from which you issue the `FLUSH TABLES` statement so that the read lock remains in effect. (If you exit the client, the lock is released.) Then take a snapshot of the data on your master server.

The easiest way to create a snapshot is to use an archiving program to make

a binary backup of the databases in your master's data directory. For example, use **tar** on Unix, or **PowerArchiver**, **WinRAR**, **WinZip**, or any similar software on Windows. To use **tar** to create an archive that includes all databases, change location into the master server's data directory, then execute this command:

```
shell> tar -cvf /tmp/mysql-snapshot.tar .
```

If you want the archive to include only a database called `this_db`, use this command instead:

```
shell> tar -cvf /tmp/mysql-snapshot.tar ./this_db
```

Then copy the archive file to the `/tmp` directory on the slave server host. On that machine, change location into the slave's data directory, and unpack the archive file using this command:

```
shell> tar -xvf /tmp/mysql-snapshot.tar
```

You may not want to replicate the `mysql` database if the slave server has a different set of user accounts from those that exist on the master. In this case, you should exclude it from the archive. You also need not include any log files in the archive, or the master `.info` or `relay-log.info` files.

While the read lock placed by `FLUSH TABLES WITH READ LOCK` is in effect, read the value of the current binary log name and offset on the master:

```
mysql > SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.003 | 73       | test         | manual,mysql      |
+-----+-----+-----+-----+
```

The `File` column shows the name of the log and `Position` shows the offset within the file. In this example, the binary log file is `mysql-bin.003` and the offset is 73. Record these values. You need them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

If the master has been running previously without binary logging enabled, the log name and position values displayed by `SHOW MASTER STATUS` or

mysqldump --master-data will be empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string (' ') and 4.

After you have taken the snapshot and recorded the log name and offset, you can re-enable write activity on the master:

```
mysql> UNLOCK TABLES;
```

If you are using InnoDB tables, ideally you should use the **InnoDB Hot Backup** tool, which takes a consistent snapshot without acquiring any locks on the master server, and records the log name and offset corresponding to the snapshot to be later used on the slave. **Hot Backup** is an additional non-free (commercial) tool that is not included in the standard MySQL distribution. See the **InnoDB Hot Backup** home page at <http://www.innodb.com/manual.php> for detailed information.

Without the **Hot Backup** tool, the quickest way to take a binary snapshot of InnoDB tables is to shut down the master server and copy the InnoDB data files, log files, and table format files (.frm files). To record the current log file name and offset, you should issue the following statements before you shut down the server:

```
mysql> FLUSH TABLES WITH READ LOCK;  
mysql> SHOW MASTER STATUS;
```

Then record the log name and the offset from the output of SHOW MASTER STATUS as was shown earlier. After recording the log name and the offset, shut down the server *without* unlocking the tables to make sure that the server goes down with the snapshot corresponding to the current log file and offset:

```
shell> mysqladmin -u root shutdown
```

An alternative that works for both MyISAM and InnoDB tables is to take an SQL dump of the master instead of a binary copy as described in the preceding discussion. For this, you can use **mysqldump --master-data** on your master and later load the SQL dump file into your slave. However, this is slower than doing a binary copy.

4. Make sure that the [mysqld] section of the my.cnf file on the master host

includes a `log-bin` option. The section should also have a `server-id=master_id` option, where *master_id* must be a positive integer value from 1 to $2^{32} - 1$. For example:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

If those options are not present, add them and restart the server. The server cannot act as a replication master unless binary logging is enabled.

Note: For the greatest possible durability and consistency in a replication setup using InnoDB with transactions, you should use `innodb_flush_log_at_trx_commit=1`, `sync_binlog=1`, and, before MySQL 5.0.3, `innodb_safe_binlog`, in the master `my.cnf` file. (`innodb_safe_binlog` is not needed from 5.0.3 on.)

5. Stop the server that is to be used as a slave and add the following lines to its `my.cnf` file:

```
[mysqld]
server-id=slave_id
```

The *slave_id* value, like the *master_id* value, must be a positive integer value from 1 to $2^{32} - 1$. In addition, it is necessary that the ID of the slave be different from the ID of the master. For example:

```
[mysqld]
server-id=2
```

If you are setting up multiple slaves, each one must have a unique `server-id` value that differs from that of the master and from each of the other slaves. Think of `server-id` values as something similar to IP addresses: These IDs uniquely identify each server instance in the community of replication partners.

If you do not specify a `server-id` value, it is set to 1 if you have not defined `master-host`; otherwise it is set to 2. Note that in the case of `server-id` omission, a master refuses connections from all slaves, and a slave refuses to connect to a master. Thus, omitting `server-id` is good only for backup with a binary log.

6. If you made a binary backup of the master server's data, copy it to the slave server's data directory before starting the slave. Make sure that the privileges on the files and directories are correct. The system account that you use to run the slave server must be able to read and write the files, just as on the master.

If you made a backup using **mysqldump**, start the slave first. The dump file is loaded in a later step.

7. Start the slave server. If it has been replicating previously, start the slave server with the `--skip-slave-start` option so that it doesn't immediately try to connect to its master. You also may want to start the slave server with the `--log-warnings` option to get more messages in the error log about problems (for example, network or connection problems). The option is enabled by default, but aborted connections are not logged to the error log unless the option value is greater than 1.
8. If you made a backup of the master server's data using **mysqldump**, load the dump file into the slave server:

```
shell> mysql -u root -p < dump_file.sql
```

9. Execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
->     MASTER_HOST='master_host_name',
->     MASTER_USER='replication_user_name',
->     MASTER_PASSWORD='replication_password',
->     MASTER_LOG_FILE='recorded_log_file_name',
->     MASTER_LOG_POS=recorded_log_position;
```

The following table shows the maximum allowable length for the string-valued options:

MASTER_HOST	60
MASTER_USER	16
MASTER_PASSWORD	32
MASTER_LOG_FILE	255

10. Start the slave threads:

```
mysql> START SLAVE;
```

After you have performed this procedure, the slave should connect to the master and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server-id` option for the master, slaves cannot connect to it.

If you have forgotten to set the `server-id` option for the slave, you get the following error in the slave's error log:

```
Warning: You should set server-id to a non-0 value if master_host is set; we will force server id to 2, but this MySQL server will not act as a slave.
```

You also find error messages in the slave's error log if it is not able to replicate for any other reason.

Once a slave is replicating, you can find in its data directory one file named `master.info` and another named `relay-log.info`. The slave uses these two files to keep track of how much of the master's binary log it has processed. Do *not* remove or edit these files unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement to change replication parameters. The slave will use the values specified in the statement to update the status files automatically.

Note: The content of `master.info` overrides some of the server options specified on the command line or in `my.cnf`. See [Section 6.8, “Replication Startup Options”](#), for more details.

Once you have a snapshot of the master, you can use it to set up other slaves by following the slave portion of the procedure just described. You do not need to take another snapshot of the master; you can use the same one for each slave.

6.5. Replication Compatibility Between MySQL Versions

The binary log format as implemented in MySQL 5.0 is considerably different from that used in previous versions. Major changes were made in MySQL 5.0.3 (for improvements to handling of character sets and `LOAD DATA INFILE`) and 5.0.4 (for improvements to handling of time zones).

We recommend using the most recent MySQL version available because replication capabilities are continually being improved. We also recommend using the same version for both the master and the slave. We recommend upgrading masters and slaves running alpha or beta versions to new (production) versions. Replication from a 5.0.3 master to a 5.0.2 slave will fail; from a 5.0.4 master to a 5.0.3 slave will also fail. In general, slaves running MySQL 5.0.x may be used with older masters (even those running MySQL 3.23, 4.0, or 4.1), but not the reverse. For more information on potential issues, see [Section 6.7, “Replication Features and Known Problems”](#).

Note: You *cannot* replicate from a master that uses a newer binary log format to a slave that uses an older format (for example, from MySQL 5.0 to MySQL 4.1.) This has significant implications for upgrading replication servers, as described in [Section 6.6, “Upgrading a Replication Setup”](#).

The preceding information pertains to replication compatibility at the protocol level. However, there can be other constraints, such as SQL-level compatibility issues. For example, a 5.0 master cannot replicate to a 4.1 slave if the replicated statements use SQL features available in 5.0 but not in 4.1. These and other issues are discussed in [Section 6.7, “Replication Features and Known Problems”](#).

6.6. Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading.

6.6.1. Upgrading Replication to 5.0

This section applies to upgrading replication from MySQL 3.23, 4.0, or 4.1 to MySQL 5.0. A 4.0 server should be 4.0.3 or newer.

When you upgrade a master to 5.0 from an earlier MySQL release series, you should first ensure that all the slaves of this master are using the same 5.0.x release. If this is not the case, you should first upgrade the slaves. To upgrade each slave, shut it down, upgrade it to the appropriate 5.0.x version, restart it, and restart replication. The 5.0 slave is able to read the old relay logs written prior to the upgrade and to execute the statements they contain. Relay logs created by the slave after the upgrade are in 5.0 format.

After the slaves have been upgraded, shut down the master, upgrade it to the same 5.0.x release as the slaves, and restart it. The 5.0 master is able to read the old binary logs written prior to the upgrade and to send them to the 5.0 slaves. The slaves recognize the old format and handle it properly. Binary logs created by the master following the upgrade are in 5.0 format. These too are recognized by the 5.0 slaves.

In other words, there are no measures to take when upgrading to MySQL 5.0, except that the slaves must be MySQL 5.0 before you can upgrade the master to 5.0. Note that downgrading from 5.0 to older versions does not work so simply: You must ensure that any 5.0 binary logs or relay logs have been fully processed, so that you can remove them before proceeding with the downgrade.

6.7. Replication Features and Known Problems

In general, replication compatibility at the SQL level requires that any features used be supported by both the master and the slave servers. If you use a feature on a master server that is available only as of a given version of MySQL, you cannot replicate to a slave that is older than that version. Such incompatibilities are likely to occur between series, so that, for example, you cannot replicate from MySQL 5.0 to 4.1. However, these incompatibilities also can occur for within-series replication. For example, the `SLEEP()` function is available in MySQL 5.0.12 and up. If you use this function on the master server, you cannot replicate to a slave server that is older than MySQL 5.0.12.

If you are planning to use replication between 5.0 and a previous version of MySQL you should consult the edition of the MySQL Reference Manual corresponding to the earlier release series for information regarding the replication characteristics of that series.

The following list provides details about what is supported and what is not. Additional InnoDB-specific information about replication is given in [Section 14.2.6.5, “InnoDB and MySQL Replication”](#).

Replication issues with regard to stored routines and triggers is described in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

- **Known issue:** In MySQL 5.0.17, the syntax for `CREATE TRIGGER` changed to include a `DEFINER` clause for specifying which access privileges to check at trigger invocation time. (See [Section 18.1, “CREATE TRIGGER Syntax”](#), for more information.) However, if you attempt to replicate from a master server older than MySQL 5.0.17 to a slave running MySQL 5.0.17 through 5.0.19, replication of `CREATE TRIGGER` statements fails on the slave with a `Definer not fully qualified` error. A workaround is to create triggers on the master using a version-specific comment embedded in each `CREATE TRIGGER` statement:

```
CREATE /*!50017 DEFINER = 'root'@'localhost' */ TRIGGER ... ;
```

`CREATE TRIGGER` statements written this way will replicate to newer slaves, which pick up the `DEFINER` clause from the comment and execute

successfully.

This slave problem is fixed as of MySQL 5.0.20.

- Replication of `AUTO_INCREMENT`, `LAST_INSERT_ID()`, and `TIMESTAMP` values is done correctly.

However, adding an `AUTO_INCREMENT` column to a table with `ALTER TABLE` might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to the table `t1`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table `t1` has columns `col1` and `col2`.

This set of statements will also produce a new table `t2` identical to `t1`, with the addition of an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE T2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important: To guarantee the same ordering on both master and slave, *all* columns of `t1` must be referenced in the `ORDER BY` clause.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

See also [Section A.7.1, “Problems with ALTER TABLE”](#).

- The `USER()`, `UUID()`, and `LOAD_FILE()` functions are replicated without

change and thus do not work reliably on the slave.

- As of MySQL 5.0.13, the `SYSDATE()` function is no longer equivalent to `NOW()`. Implications are that `SYSDATE()` is not replication-safe because it is not affected by `SET TIMESTAMP` statements in the binary log and is non-deterministic. To avoid this, you can start the server with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`.
- User privileges are replicated only if the `mysql` database is replicated. That is, the `GRANT`, `REVOKE`, `SET PASSWORD`, `CREATE USER`, and `DROP USER` statements take effect on the slave only if the replication setup includes the `mysql` database.

If you're replicating all databases, but don't want statements that affect user privileges to be replicated, set up the slave to not replicate the `mysql` database, using the `--replicate-wild-ignore-table=mysql.%` option. The slave will recognize that issuing privilege-related SQL statements won't have an effect, and thus not execute those statements.

- The `GET_LOCK()`, `RELEASE_LOCK()`, `IS_FREE_LOCK()`, and `IS_USED_LOCK()` functions that handle user-level locks are replicated without the slave knowing the concurrency context on master. Therefore, these functions should not be used to insert into a master's table because the content on the slave would differ. (For example, do not issue a statement such as `INSERT INTO mytable VALUES(GET_LOCK(...))`.)
- The `FOREIGN_KEY_CHECKS`, `SQL_MODE`, `UNIQUE_CHECKS`, and `SQL_AUTO_IS_NULL` variables are all replicated in MySQL 5.0. The `storage_engine` system variable (also known as `table_type`) is not yet replicated, which is a good thing for replication between different storage engines.
- Starting from MySQL 5.0.3 (master and slave), replication works even if the master and slave have different global character set variables. Starting from MySQL 5.0.4 (master and slave), replication works even if the master and slave have different global time zone variables.
- The following applies to replication between MySQL servers that use different character sets:

1. If the master uses MySQL 4.1, you must *always* use the same *global* character set and collation on the master and the slave, regardless of the MySQL version running on the slave. (These are controlled by the `--character-set-server` and `--collation-server` options.) Otherwise, you may get duplicate-key errors on the slave, because a key that is unique in the master character set might not be unique in the slave character set. Note that this is not a cause for concern when master and slave are both MySQL 5.0 or later.
 2. If the master is older than MySQL 4.1.3, the character set of any client should never be made different from its global value because this character set change is not known to the slave. In other words, clients should not use `SET NAMES`, `SET CHARACTER SET`, and so forth. If both the master and the slave are 4.1.3 or newer, clients can freely set session values for character set variables because these settings are written to the binary log and so are known to the slave. That is, clients can use `SET NAMES` or `SET CHARACTER SET` or can set variables such as `collation_client` or `collation_server`. However, clients are prevented from changing the *global* value of these variables; as stated previously, the master and slave must always have identical global character set values.
 3. If you have databases on the master with character sets that differ from the global `character_set_server` value, you should design your `CREATE TABLE` statements so that tables in those databases do not implicitly rely on the database default character set (see Bug #2326). A good workaround is to state the character set and collation explicitly in `CREATE TABLE` statements.
- If the master uses MySQL 4.1, the same system time zone should be set for both master and slave. Otherwise some statements will not be replicated properly, such as statements that use the `NOW()` or `FROM_UNIXTIME()` functions. You can set the time zone in which MySQL server runs by using the `--timezone=timezone_name` option of the `mysqld_safe` script or by setting the `TZ` environment variable. Both master and slave should also have the same default connection time zone setting; that is, the `--default-time-zone` parameter should have the same value for both master and slave. Note that this is not necessary when the master is MySQL 5.0 or later.

- `CONVERT_TZ(..., ..., @@global.time_zone)` is not properly replicated. `CONVERT_TZ(..., ..., @@session.time_zone)` is properly replicated only if the master and slave are from MySQL 5.0.4 or newer.
- Session variables are not replicated properly when used in statements that update tables. For example, `SET MAX_JOIN_SIZE=1000` followed by `INSERT INTO mytable VALUES(@@MAX_JOIN_SIZE)` will not insert the same data on the master and the slave. This does not apply to the common sequence of `SET TIME_ZONE=...` followed by `INSERT INTO mytable VALUES(CONVERT_TZ(..., ..., @@time_zone))`, which replicates correctly as of MySQL 5.0.4.
- It is possible to replicate transactional tables on the master using non-transactional tables on the slave. For example, you can replicate an InnoDB master table as a MyISAM slave table. However, if you do this, there are problems if the slave is stopped in the middle of a `BEGIN/COMMIT` block because the slave restarts at the beginning of the `BEGIN` block.
- Update statements that refer to user-defined variables (that is, variables of the form `@var_name`) are replicated correctly in MySQL 5.0. However, this is not true for versions prior to 4.1. Note that user variable names are case insensitive starting in MySQL 5.0. You should take this into account when setting up replication between MySQL 5.0 and older versions.
- Slaves can connect to masters using SSL.
- Views are always replicated to slaves. Views are filtered by their own name, not by the tables they refer to. This means that a view can be replicated to the slave even if the view contains a table that would normally be filtered out by `replication-ignore-table` rules. Care should therefore be taken to ensure that views do not replicate table data that would normally be filtered for security reasons.
- In MySQL 5.0 (starting from 5.0.3), there is a global system variable `slave_transaction_retries`: If the replication slave SQL thread fails to execute a transaction because of an InnoDB deadlock or because it exceeded the InnoDB `innodb_lock_wait_timeout` or the NDBCluster `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout` value, the transaction automatically retries

`slave_transaction_retries` times before stopping with an error. The default value is 10. Starting from MySQL 5.0.4, the total retry count can be seen in the output of `SHOW STATUS`; see [Section 5.2.4, “Server Status Variables”](#).

- If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the master server, the table option is also used on the slave. This can cause problems if no corresponding directory exists in the slave host filesystem or if it exists but is not accessible to the slave server. MySQL supports an `sql_mode` option called `NO_DIR_IN_CREATE`. If the slave server is run with this SQL mode enabled, it ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that MyISAM data and index files are created in the table's database directory.
- It is possible for the data on the master and slave to become different if a statement is designed in such a way that the data modification is non-deterministic; that is, left to the will of the query optimizer. (This is in general not a good practice, even outside of replication.) For a detailed explanation of this issue, see [Section A.8.1, “Open Issues in MySQL”](#).
- Using `LOAD TABLE FROM MASTER` where the master is running MySQL 4.1 and the slave is running MySQL 5.0 may corrupt the table data, and is not supported. (Bug #16261)
- *The following applies only if either the master or the slave is running MySQL version 5.0.3 or older:* If on the master a `LOAD DATA INFILE` is interrupted (integrity constraint violation, killed connection, and so on), the slave skips the `LOAD DATA INFILE` entirely. This means that if this command permanently inserted or updated table records before being interrupted, these modifications are not replicated to the slave.
- Some forms of the `FLUSH` statement are not logged because they could cause problems if replicated to a slave: `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK`. For a syntax example, see [Section 13.5.5.2, “FLUSH Syntax”](#). The `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements are written to the binary log and thus replicated to slaves. This is not normally a problem because these statements do not modify table data. However, this can cause

difficulties under certain circumstances. If you replicate the privilege tables in the `mysql` database and update those tables directly without using `GRANT`, you must issue a `FLUSH PRIVILEGES` on the slaves to put the new privileges into effect. In addition, if you use `FLUSH TABLES` when renaming a MyISAM table that is part of a MERGE table, you must issue `FLUSH TABLES` manually on the slaves. These statements are written to the binary log unless you specify `NO_WRITE_TO_BINLOG` or its alias `LOCAL`.

- MySQL supports only one master and many slaves. In the future we plan to add a voting algorithm for changing the master automatically in the event of problems with the current master. We also plan to introduce agent processes to help perform load balancing by sending `SELECT` queries to different slaves.
- When a server shuts down and restarts, its `MEMORY` (HEAP tables become empty. The master replicates this effect to slaves as follows: The first time that the master uses each `MEMORY` table after startup, it logs an event that notifies the slaves that the table needs to be emptied by writing a `DELETE` statement for that table to the binary log. See [Section 14.4, “The MEMORY \(HEAP\) Storage Engine”](#), for more information.
- Temporary tables are replicated except in the case where you shut down the slave server (not just the slave threads) and you have replicated temporary tables that are used in updates that have not yet been executed on the slave. If you shut down the slave server, the temporary tables needed by those updates are no longer available when the slave is restarted. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use the following procedure:
 1. Issue a `STOP SLAVE` statement.
 2. Use `SHOW STATUS` to check the value of the `Slave_open_temp_tables` variable.
 3. If the value is 0, issue a **`mysqladmin shutdown`** command to stop the slave.
 4. If the value is not 0, restart the slave threads with `START SLAVE`.
 5. Repeat the procedure later until the `Slave_open_temp_tables` variable

is 0 and you can stop the slave.

- The syntax for multiple-table DELETE statements that use table aliases changed between MySQL 4.0 and 4.1. In MySQL 4.0, you should use the true table name to refer to any table from which rows should be deleted:

```
DELETE test FROM test AS t1, test2 WHERE ...
```

In MySQL 4.1, you must use the alias:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

If you use such DELETE statements, the change in syntax means that a 4.0 master cannot replicate to 4.1 (or higher) slaves.

- It is safe to connect servers in a circular master/slave relationship if you use the `--log-slave-updates` option. That means that you can create a setup such as this:

```
A -> B -> C -> A
```

However, many statements do not work correctly in this kind of setup unless your client code is written to take care of the potential problems that can occur from updates that occur in different sequence on different servers.

Server IDs are encoded in binary log events, so server A knows when an event that it reads was originally created by itself and does not execute the event (unless server A was started with the `--replicate-same-server-id` option, which is meaningful only in rare cases). Thus, there are no infinite loops. This type of circular setup works only if you perform no conflicting updates between the tables. In other words, if you insert data in both A and C, you should never insert a row in A that may have a key that conflicts with a row inserted in C. You should also not update the same rows on two servers if the order in which the updates are applied is significant.

- If a statement on a slave produces an error, the slave SQL thread terminates, and the slave writes a message to its error log. You should then connect to the slave manually and determine the cause of the problem. (`SHOW SLAVE STATUS` is useful for this.) Then fix the problem (for example, you might need to create a non-existent table) and run `START SLAVE`.

- It is safe to shut down a master server and restart it later. When a slave loses its connection to the master, the slave tries to reconnect immediately and retries periodically if that fails. The default is to retry every 60 seconds. This may be changed with the `--master-connect-retry` option. A slave also is able to deal with network connectivity outages. However, the slave notices the network outage only after receiving no data from the master for `slave_net_timeout` seconds. If your outages are short, you may want to decrease `slave_net_timeout`. See [Section 5.2.2, “Server System Variables”](#).
- Shutting down the slave (cleanly) is also safe because it keeps track of where it left off. Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the system went down. Your system fault tolerance is greatly increased if you have a good uninterruptible power supply. Unclean shutdowns of the master may cause inconsistencies between the content of tables and the binary log in master; this can be avoided by using InnoDB tables and the `--innodb-safe-binlog` option on the master. See [Section 5.12.3, “The Binary Log”](#).

Note: `--innodb-safe-binlog` is unneeded as of MySQL 5.0.3, having been made obsolete by the introduction of XA transaction support.

- A crash on the master side can result in the master's binary log having a final position less than the most recent position read by the slave, due to the master's binary log file not being flushed. This can cause the slave not to be able to replicate when the master comes back up. Setting `sync_binlog=1` in the master `my.cnf` file helps to minimize this problem because it causes the master to flush its binary log more frequently.
- Due to the non-transactional nature of MyISAM tables, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the master, the slave thread exits and waits for the database administrator to decide what to do about it unless the error code is legitimate and execution of the statement results in the same error code on the slave. If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the `--slave-skip-errors` option.

- If you update transactional tables from non-transactional tables inside a BEGIN/COMMIT sequence, updates to the binary log may be out of synchrony with table states if the non-transactional table is updated before the transaction commits. This occurs because the transaction is written to the binary log only when it is committed.
- In situations where transactions mix updates to transactional and non-transactional tables, the order of statements in the binary log is correct, and all needed statements are written to the binary log even in case of a ROLLBACK. However, when a second connection updates the non-transactional table before the first connection's transaction is complete, statements can be logged out of order, because the second connection's update is written immediately after it is performed, regardless of the state of the transaction being performed by the first connection.
- Floating-point values are approximate, so comparisons involving them are inexact. This is true for operations that use floating-point values explicitly, or values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on master and slave servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See [Section 12.1.2, “Type Conversion in Expression Evaluation”](#), and [Section A.5.8, “Problems with Floating-Point Comparisons”](#).

6.8. Replication Startup Options

This section describes the options that you can use on slave replication servers. You can specify these options either on the command line or in an option file.

On the master and each slave, you must use the `server-id` option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to $2^{32} - 1$, and each ID must be different from every other ID. Example: `server-id=3`

Options that you can use on the master server for controlling binary logging are described in [Section 5.12.3, “The Binary Log”](#).

Some slave server replication options are handled in a special way, in the sense that each is ignored if a `master.info` file exists when the slave starts and contains a value for the option. The following options are handled this way:

- `--master-host`
- `--master-user`
- `--master-password`
- `--master-port`
- `--master-connect-retry`
- `--master-ssl`
- `--master-ssl-ca`
- `--master-ssl-capath`
- `--master-ssl-cert`
- `--master-ssl-cipher`
- `--master-ssl-key`

The `master.info` file format in MySQL 5.0 includes values corresponding to the SSL options. In addition, the file format includes as its first line the number of lines in the file. (See [Section 6.3.4, “Replication Relay and Status Files”](#).) If you upgrade an older server (before MySQL 4.1.1) to a newer version, the new server upgrades the `master.info` file to the new format automatically when it starts. However, if you downgrade a newer server to an older version, you should remove the first line manually before starting the older server for the first time.

If no `master.info` file exists when the slave server starts, it uses the values for those options that are specified in option files or on the command line. This occurs when you start the server as a replication slave for the very first time, or when you have run `RESET SLAVE` and then have shut down and restarted the slave.

If the `master.info` file exists when the slave server starts, the server uses its contents and ignores any options that correspond to the values listed in the file. Thus, if you start the slave server with different values of the startup options that correspond to values in the `master.info` file, the different values have no effect, because the server continues to use the `master.info` file. To use different values, you must either restart after removing the `master.info` file or (preferably) use the `CHANGE MASTER TO` statement to reset the values while the slave is running.

Suppose that you specify this option in your `my.cnf` file:

```
[mysqld]
master-host=some_host
```

The first time you start the server as a replication slave, it reads and uses that option from the `my.cnf` file. The server then records the value in the `master.info` file. The next time you start the server, it reads the master host value from the `master.info` file only and ignores the value in the option file. If you modify the `my.cnf` file to specify a different master host of *some_other_host*, the change still has no effect. You should use `CHANGE MASTER TO` instead.

Because the server gives an existing `master.info` file precedence over the startup options just described, you might prefer not to use startup options for these values at all, and instead specify them by using the `CHANGE MASTER TO` statement. See [Section 13.6.2.1, “CHANGE MASTER TO Syntax”](#).

This example shows a more extensive use of startup options to configure a slave server:

```
[mysqld]
server-id=2
master-host=db-master.mycompany.com
master-port=3306
master-user=pertinax
master-password=freitag
master-connect-retry=60
report-host=db-slave.mycompany.com
```

The following list describes startup options for controlling replication. Many of these options can be reset while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the slave server starts.

- `--log-slave-updates`

Normally, a slave does not log to its own binary log any updates that are received from a master server. This option tells the slave to log the updates performed by its SQL thread to its own binary log. For this option to have any effect, the slave must also be started with the `--log-bin` option to enable binary logging. `--log-slave-updates` is used when you want to chain replication servers. For example, you might want to set up replication servers using this arrangement:

```
A -> B -> C
```

Here, A serves as the master for the slave B, and B serves as the master for the slave C. For this to work, B must be both a master *and* a slave. You must start both A and B with `--log-bin` to enable binary logging, and B with the `--log-slave-updates` option so that updates received from A are logged by B to its binary log.

- `--log-warnings[=level]`

This option causes a server to print more messages to the error log about what it is doing. With respect to replication, the server generates warnings that it succeeded in reconnecting after a network/connection failure, and informs you as to how each slave thread started. This option is enabled by default; to disable it, use `--skip-log-warnings`. Aborted connections are

not logged to the error log unless the value is greater than 1.

- `--master-connect-retry=seconds`

The number of seconds that the slave thread sleeps before trying to reconnect to the master in case the master goes down or the connection is lost. The value in the `master.info` file takes precedence if it can be read. If not set, the default is 60.

- `--master-host=host_name`

The hostname or IP number of the master replication server. The value in `master.info` takes precedence if it can be read. If no master host is specified, the slave thread does not start.

- `--master-info-file=file_name`

The name to use for the file in which the slave records information about the master. The default name is `master.info` in the data directory.

- `--master-password=password`

The password of the account that the slave thread uses for authentication when it connects to the master. The value in the `master.info` file takes precedence if it can be read. If not set, an empty password is assumed.

- `--master-port=port_number`

The TCP/IP port number that the master is listening on. The value in the `master.info` file takes precedence if it can be read. If not set, the compiled-in setting is assumed (normally 3306).

- `--master-retry-count=count`

The number of times that the slave tries to connect to the master before giving up.

- `--master-ssl, --master-ssl-ca=file_name, --master-ssl-capath=directory_name, --master-ssl-cert=file_name, --master-ssl-cipher=cipher_list, --master-ssl-key=file_name`

These options are used for setting up a secure replication connection to the master server using SSL. Their meanings are the same as the corresponding `--ssl`, `--ssl-ca`, `--ssl-capath`, `--ssl-cert`, `--ssl-cipher`, `--ssl-key` options that are described in [Section 5.9.7.3, “SSL Command Options”](#). The values in the `master.info` file take precedence if they can be read.

- `--master-user=user_name`

The username of the account that the slave thread uses for authentication when it connects to the master. This account must have the `REPLICATION SLAVE` privilege. The value in the `master.info` file takes precedence if it can be read. If the master username is not set, the name `test` is assumed.

- `--max-relay-log-size=size`

The size at which the server rotates relay log files automatically. For more information, see [Section 6.3.4, “Replication Relay and Status Files”](#).

- `--read-only`

Cause the slave to allow no updates except from slave threads or from users having the `SUPER` privilege. This enables you to ensure that a slave server accepts no updates from clients. As of MySQL 5.0.16, this option does not apply to `TEMPORARY` tables.

- `--relay-log=file_name`

The name for the relay log. The default name is `host_name-relay-bin.nnnnnn`, where *host_name* is the name of the slave server host and *nnnnnn* indicates that relay logs are created in numbered sequence. You can specify the option to create hostname-independent relay log names, or if your relay logs tend to be big (and you don't want to decrease `max_relay_log_size`) and you need to put them in some area different from the data directory, or if you want to increase speed by balancing load between disks.

- `--relay-log-index=file_name`

The name to use for the relay log index file. The default name is `host_name-relay-bin.index` in the data directory, where *host_name* is the

name of the slave server.

- `--relay-log-info-file=file_name`

The name to use for the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory.

- `--relay-log-purge={0|1}`

Disable or enable automatic purging of relay logs as soon as they are not needed any more. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = N`.

- `--relay-log-space-limit=size`

This option places an upper limit on the total size in bytes of all relay logs on the slave. A value of 0 means “no limit.” This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs, because not doing so would cause a deadlock. You should not set `--relay-log-space-limit` to less than twice the value of `--max-relay-log-size` (or `--max-binlog-size` if `--max-relay-log-size` is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to temporarily ignore `--relay-log-space-limit`.

- `--replicate-do-db=db_name`

Tell the slave to restrict replication to statements where the default database (that is, the one selected by `USE`) is `db_name`. To specify more than one database, use this option multiple times, once for each database. Note that this does not replicate cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` while having selected a different database or no database.

An example of what does not work as you might expect: If the slave is started with `--replicate-do-db=sales` and you issue the following statements on the master, the UPDATE statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table DELETE statements or multiple-table UPDATE statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

If you need cross-database updates to work, use `--replicate-wild-do-table=db_name.%` instead. See [Section 6.9, “How Servers Evaluate Replication Rules”](#).

- `--replicate-do-table=db_name.tbl_name`

Tell the slave thread to restrict replication to the specified table. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-do-db`. See [Section 6.9, “How Servers Evaluate Replication Rules”](#).

- `--replicate-ignore-db=db_name`

Tells the slave to not replicate any statement where the default database (that is, the one selected by USE) is `db_name`. To specify more than one database to ignore, use this option multiple times, once for each database. You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See [Section 6.9, “How Servers Evaluate Replication Rules”](#).

An example of what does not work as you might expect: If the slave is started with `--replicate-ignore-db=sales` and you issue the following statements on the master, the UPDATE statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

If you need cross-database updates to work, use `--replicate-wild-ignore-table=db_name.%` instead. See [Section 6.9, “How Servers Evaluate Replication Rules”](#).

- `--replicate-ignore-table=db_name.tbl_name`

Tells the slave thread to not replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-ignore-db`. See [Section 6.9, “How Servers Evaluate Replication Rules”](#).

- `--replicate-rewrite-db=from_name->to_name`

Tells the slave to translate the default database (that is, the one selected by `USE`) to *to_name* if it was *from_name* on the master. Only statements involving tables are affected (not statements such as `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), and only if *from_name* is the default database on the master. This does not work for cross-database updates. The database name translation is done *before* the `--replicate-*` rules are tested.

If you use this option on the command line and the ‘>’ character is special to your command interpreter, quote the option value. For example:

```
shell> mysqld --replicate-rewrite-db="olddb->newdb"
```

- `--replicate-same-server-id`

To be used on slave servers. Usually you should use the default setting of 0, to prevent infinite loops caused by circular replication. If set to 1, the slave does not skip events having its own server ID. Normally, this is useful only in rare configurations. Cannot be set to 1 if `--log-slave-updates` is used. Note that by default the slave I/O thread does not even write binary log events to the relay log if they have the slave's server id (this optimization helps save disk usage). So if you want to use `--replicate-same-server-id`, be sure to start the slave with this option before you make the slave read its own events that you want the slave SQL thread to execute.

- `--replicate-wild-do-table=db_name.tbl_name`

Tells the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the ‘%’ and ‘_’ wildcard characters, which have the same meaning as for the LIKE pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See [Section 6.9, “How Servers Evaluate Replication Rules”](#).

Example: `--replicate-wild-do-table=foo%.bar%` replicates only updates that use a table where the database name starts with foo and the table name starts with bar.

If the table name pattern is %, it matches any table name and the option also applies to database-level statements (CREATE DATABASE, DROP DATABASE, and ALTER DATABASE). For example, if you use `--replicate-wild-do-table=foo%.`, database-level statements are replicated if the database name matches the pattern foo%.

To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named my_own%db, but not replicate tables from the my1ownAABCdb database, you should escape the ‘_’ and ‘%’ characters like this: `--replicate-wild-do-table=my_own\%db`. If you're using the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the **bash** shell, you would need to type `--replicate-wild-do-table=my_own\\%db`.

- `--replicate-wild-ignore-table=db_name.tbl_name`

Tells the slave thread not to replicate a statement where any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See [Section 6.9, “How Servers Evaluate Replication Rules”](#).

Example: `--replicate-wild-ignore-table=foo%.bar%` does not replicate updates that use a table where the database name starts with foo and the

table name starts with bar.

For information about how matching works, see the description of the `--replicate-wild-do-table` option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` as well.

- `--report-host=slave_name`

The hostname or IP number of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master. Note that it is not sufficient for the master to simply read the IP number of the slave from the TCP/IP socket after the slave connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

- `--report-port=slave_port_num`

The TCP/IP port number for connecting to the slave, to be reported to the master during slave registration. Set this only if the slave is listening on a non-default port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, do not use this option.

- `--skip-slave-start`

Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave_compressed_protocol={0|1}`

If this option is set to 1, use compression for the slave/master protocol if both the slave and the master support it.

- `--slave-load-tmpdir=file_name`

The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` system variable. When the slave SQL thread replicates a `LOAD DATA INFILE` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads

these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some filesystem that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `--relay-log` option to place the relay logs in that filesystem.

The directory specified by this option should be located in a disk-based filesystem (not a memory-based filesystem) because the temporary files used to replicate `LOAD DATA INFILE` must survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process.

- `--slave-net-timeout=seconds`

The number of seconds to wait for more data from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `--master-connect-retry` option.

- `--slave-skip-errors=[err_code1,err_code2,...|all]`

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This option tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of synchrony with the master, with you having no idea why this has occurred.

For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`.

[Appendix B, Error Codes and Messages](#), lists server error codes.

You can also (but should not) use the very non-recommended value of `all` to cause the slave to ignore all error messages and keeps going regardless of

what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the slave's data is not anywhere close to what it is on the master. *You have been warned.*

Examples:

```
--slave-skip-errors=1062,1053  
--slave-skip-errors=all
```

6.9. How Servers Evaluate Replication Rules

If a master server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all slaves and each slave determines whether to execute it or ignore it.

On the master side, decisions about which statements to log are based on the `--binlog-do-db` and `--binlog-ignore-db` options that control binary logging. For a description of the rules that servers use in evaluating these options, see [Section 5.12.3, “The Binary Log”](#).

On the slave side, decisions about whether to execute or ignore statements received from the master are made according to the `--replicate-*` options that the slave was started with. (See [Section 6.8, “Replication Startup Options”](#).) The slave evaluates these options using the following procedure, which first checks the database-level options and then the table-level options.

In the simplest case, when there are no `--replicate-*` options, the procedure yields the result that the slave executes all statements that it receives from the master. Otherwise, the result depends on the particular options given. In general, to make it easier to determine what effect an option set will have, it is recommended that you avoid mixing “do” and “ignore” options, or wildcard and non-wildcard options.

Stage 1. Check the database options.

At this stage, the slave checks whether there are any `--replicate-do-db` or `--replicate-ignore-db` options that specify database-specific conditions:

- *No*: Permit the statement and proceed to the table-checking stage.
- *Yes*: Test the options using the same rules as for the `--binlog-do-db` and `--binlog-ignore-db` options to determine whether to permit or ignore the statement. What is the result of the test?
 - *Permit*: Do not execute the statement immediately. Defer the decision and proceed to the table-checking stage.

- *Ignore*: Ignore the statement and exit.

This stage can permit a statement for further option-checking, or cause it to be ignored. However, statements that are permitted at this stage are not actually executed yet. Instead, they pass to the following stage that checks the table options.

Stage 2. Check the table options.

First, as a preliminary condition, the slave checks whether the statement occurs within a stored function or (prior to MySQL 5.0.12) a stored procedure. If so, execute the statement and exit. (Stored procedures are exempt from this test as of MySQL 5.0.12 because procedure logging occurs at the level of statements that are executed within the routine rather than at the CALL level.)

Next, the slave checks for table options and evaluates them. If the server reaches this point, it executes all statements if there are no table options. If there are “do” table options, the statement must match one of them if it is to be executed; otherwise, it is ignored. If there are any “ignore” options, all statements are executed except those that match any ignore option. The following steps describe how this evaluation occurs in more detail.

1. Are there any --replicate-* -table options?

- *No*: There are no table restrictions, so all statements match. Execute the statement and exit.
- *Yes*: There are table restrictions. Evaluate the tables to be updated against them. There might be multiple tables to update, so loop through the following steps for each table looking for a matching option (first the non-wild options, and then the wild options). Only tables that are to be updated are compared to the options. For example, if the statement is `INSERT INTO sales SELECT * FROM prices`, only sales is compared to the options). If several tables are to be updated (multiple-table statement), the first table that matches “do” or “ignore” wins. That is, the server checks the first table against the options. If no decision could be made, it checks the second table against the options, and so on.

2. Are there any --replicate-do-table options?

- *No*: Proceed to the next step.
- *Yes*: Does the table match any of them?
 - *No*: Proceed to the next step.
 - *Yes*: Execute the statement and exit.

3. Are there any `--replicate-ignore-table` options?

- *No*: Proceed to the next step.
- *Yes*: Does the table match any of them?
 - *No*: Proceed to the next step.
 - *Yes*: Ignore the statement and exit.

4. Are there any `--replicate-wild-do-table` options?

- *No*: Proceed to the next step.
- *Yes*: Does the table match any of them?
 - *No*: Proceed to the next step.
 - *Yes*: Execute the statement and exit.

5. Are there any `--replicate-wild-ignore-table` options?

- *No*: Proceed to the next step.
- *Yes*: Does the table match any of them?
 - *No*: Proceed to the next step.
 - *Yes*: Ignore the statement and exit.

6. No `--replicate-*-table` option was matched. Is there another table to test against these options?

- *No*: We have now tested all tables to be updated and could not match any option. Are there `--replicate-do-table` or `--replicate-wild-do-table` options?
 - *No*: There were no “do” table options, so no explicit “do” match is required. Execute the statement and exit.
 - *Yes*: There were “do” table options, so the statement is executed only with an explicit match to one of them. Ignore the statement and exit.
- *Yes*: Loop.

Examples:

- No `--replicate-*` options at all

The slave executes all statements that it receives from the master.

- `--replicate-*-db` options, but no table options

The slave permits or ignores statements using the database options. Then it executes all statements permitted by those options because there are no table restrictions.

- `--replicate-*-table` options, but no database options

All statements are permitted at the database-checking stage because there are no database conditions. The slave executes or ignores statements based on the table options.

- A mix of database and table options

The slave permits or ignores statements using the database options. Then it evaluates all statements permitted by those options according to the table options. In some cases, this process can yield what might seem a counterintuitive result. Consider the following set of options:

```
[mysqld]
replicate-do-db      = db1
replicate-do-table  = db2.mytbl2
```

Suppose that db1 is the default database and the slave receives this statement:

```
INSERT INTO mytbl1 VALUES(1,2,3);
```

The database is db1, which matches the --replicate-do-db option at the database-checking stage. The algorithm then proceeds to the table-checking stage. If there were no table options, the statement would be executed. However, because the options include a “do” table option, the statement must match if it is to be executed. The statement does not match, so it is ignored. (The same would happen for any table in db1.)

6.10. Replication FAQ

Q: How do I configure a slave if the master is running and I do not want to stop it?

A: There are several possibilities. If you have taken a snapshot backup of the master at some point and recorded the binary log filename and offset (from the output of `SHOW MASTER STATUS`) corresponding to the snapshot, use the following procedure:

1. Make sure that the slave is assigned a unique server ID.
2. Execute the following statement on the slave, filling in appropriate values for each option:

```
mysql> CHANGE MASTER TO
->     MASTER_HOST='master_host_name',
->     MASTER_USER='master_user_name',
->     MASTER_PASSWORD='master_pass',
->     MASTER_LOG_FILE='recorded_log_file_name',
->     MASTER_LOG_POS=recorded_log_position;
```

3. Execute `START SLAVE` on the slave.

If you do not have a backup of the master server, here is a quick procedure for creating one. All steps should be performed on the master host.

1. Issue this statement to acquire a global read lock:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

2. With the lock still in place, execute this command (or a variation of it):

```
shell> tar zcf /tmp/backup.tar.gz /var/lib/mysql
```

3. Issue this statement and record the output, which you will need later:

```
mysql> SHOW MASTER STATUS;
```

4. Release the lock:

```
mysql> UNLOCK TABLES;
```

An alternative to using the preceding procedure to make a binary copy is to make an SQL dump of the master. To do this, you can use **mysqldump --master-data** on your master and later load the SQL dump into your slave. However, this is slower than making a binary copy.

Regardless of which of the two methods you use, afterward follow the instructions for the case when you have a snapshot and have recorded the log filename and offset. You can use the same snapshot to set up several slaves. Once you have the snapshot of the master, you can wait to set up a slave as long as the binary logs of the master are left intact. The two practical limitations on the length of time you can wait are the amount of disk space available to retain binary logs on the master and the length of time it takes the slave to catch up.

You can also use `LOAD DATA FROM MASTER`. This is a convenient statement that transfers a snapshot to the slave and adjusts the log filename and offset all at once. Be warned, however, that it works only for `MyISAM` tables and it may hold a read lock for a long time. It is not yet implemented as efficiently as we would like. If you have large tables, the preferred method is still to make a binary snapshot on the master server after executing `FLUSH TABLES WITH READ LOCK`.

Q: Does the slave need to be connected to the master all the time?

A: No, it does not. The slave can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a master/slave relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the slave is not guaranteed to be in synchrony with the master unless you take some special measures.

Q: How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?

A: You can read the `Seconds_Behind_Master` column in `SHOW SLAVE STATUS`. See [Section 6.3, “Replication Implementation Details”](#).

When the slave SQL thread executes an event read from the master, it modifies its own time to the event timestamp. (This is why `TIMESTAMP` is well replicated.) In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds

displayed for the slave SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. You can use this to determine the date of the last replicated event. Note that if your slave has been disconnected from the master for one hour, and then reconnects, you may immediately see `Time` values like 3600 for the slave SQL thread in `SHOW PROCESSLIST`. This is because the slave is executing statements that are one hour old.

Q: How do I force the master to block updates until the slave catches up?

A: Use the following procedure:

1. On the master, execute these statements:

```
mysql> FLUSH TABLES WITH READ LOCK;  
mysql> SHOW MASTER STATUS;
```

Record the replication coordinates (the log filename and offset) from the output of the `SHOW` statement.

2. On the slave, issue the following statement, where the arguments to the `MASTER_POS_WAIT()` function are the replication coordinate values obtained in the previous step:

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);
```

The `SELECT` statement blocks until the slave reaches the specified log file and offset. At that point, the slave is in synchrony with the master and the statement returns.

3. On the master, issue the following statement to allow the master to begin processing updates again:

```
mysql> UNLOCK TABLES;
```

Q: What issues should I be aware of when setting up two-way replication?

A: MySQL replication currently does not support any locking protocol between master and slave to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-master 1, and in the meantime, before it propagates to co-master 2, client B could make

an update to co-master 2 that makes the update of client A work differently than it did on co-master 1. Thus, when the update of client A makes it to co-master 2, it produces tables that are different from what you have on co-master 1, even after all the updates from co-master 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention, because the updates originating on another server are serialized in one slave thread. Even this benefit might be offset by network delays.

Q: How can I use replication to improve performance of my system?

A: You should set up one server as the master and direct all writes to it. Then configure as many slaves as you have the budget and rackspace for, and distribute the reads among the master and the slaves. You can also start the slaves with the `--skip-innodb`, `--skip-bdb`, `--low-priority-updates`, and `--delay-key-write=ALL` options to get speed improvements on the slave end. In this case, the slave uses non-transactional MyISAM tables instead of InnoDB and BDB tables to get more speed by eliminating transactional overhead.

Q: What should I do to prepare client code in my own applications to use performance-enhancing replication?

A: If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to clean it up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`
- `safe_reader_connect()`

- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions; for example, to log how long each statement took, or which statement among those issued gave you an error.

If you have written a lot of code, you may want to automate the conversion task by using the **replace** utility that comes with standard MySQL distributions, or write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

Q: When and how much can MySQL replication improve the performance of my system?

A: MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-master/multiple-slave setup, you can scale the system by adding more slaves until you either run out of network bandwidth, or your update load grows to the point that the master cannot handle it.

To determine how many slaves you can use before the added benefits begin to level out, and how much you can improve performance of your site, you need to know your query patterns, and to determine empirically by benchmarking the relationship between the throughput for reads (reads per second, or reads) and for writes (writes) on a typical master and a typical slave. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that reads is $1200 - 2 \times \text{writes}$. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Let us suppose that the master and each slave have the same capacity, and that we have one master and N slaves. Then we have for each server (master or slave):

$$\text{reads} = 1200 - 2 \times \text{writes}$$

$$\text{reads} = 9 \times \text{writes} / (N + 1) \text{ (reads are split, but writes go to all servers)}$$

$$9 \times \text{writes} / (N + 1) + 2 \times \text{writes} = 1200$$

$$\text{writes} = 1200 / (2 + 9/(N+1))$$

The last equation indicates the maximum number of writes for N slaves, given a maximum possible read rate of 1,200 per minute and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If $N = 0$ (which means we have no replication), our system can handle about $1200/11 = 109$ writes per second.
- If $N = 1$, we get up to 184 writes per second.
- If $N = 8$, we get up to 400 writes per second.
- If $N = 17$, we get up to 480 writes per second.
- Eventually, as N approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

Note that these computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what will happen on your system if you add N replication slaves. However, answering the following questions should help you decide

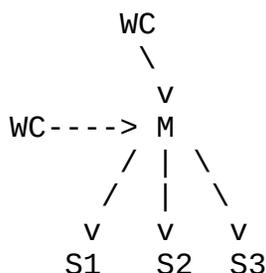
whether and by how much replication will improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many slaves do you have bandwidth available on your network?

Q: How can I use replication to provide redundancy or high availability?

A: With the currently available features, you would have to set up a master and a slave (or several slaves), and to write a script that monitors the master to check whether it is up. Then instruct your applications and the slaves to change master in case of failure. Some suggestions:

- To tell a slave to change its master, use the `CHANGE MASTER TO` statement.
- A good way to keep your applications informed as to the location of the master is by having a dynamic DNS entry for the master. With `bind` you can use `nsupdate` to dynamically update your DNS.
- Run your slaves with the `--log-bin` option and without `--log-slave-updates`. In this way, the slave is ready to become a master as soon as you issue `STOP SLAVE; RESET MASTER, and CHANGE MASTER TO` statement on the other slaves. For example, assume that you have the following setup:



In this diagram, M means the master, S the slaves, WC the clients issuing database writes and reads; clients that issue only database reads are not represented, because they need not switch. S1, S2, and S3 are slaves running with `--log-bin` and without `--log-slave-updates`. Because updates received by a slave from the master are not logged in the binary log unless `--log-slave-updates` is specified, the binary log on each slave is empty

initially. If for some reason M becomes unavailable, you can pick one of the slaves to become the new master. For example, if you pick S1, all WC should be redirected to S1, which will log updates to its binary log. S2 and S3 should then replicate from S1.

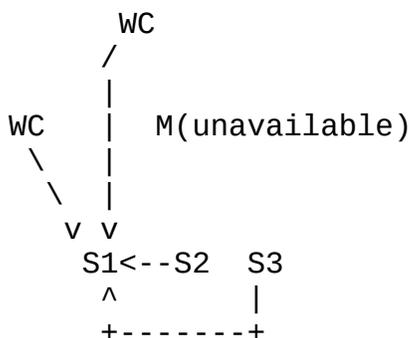
The reason for running the slave without `--log-slave-updates` is to prevent slaves from receiving updates twice in case you cause one of the slaves to become the new master. Suppose that S1 has `--log-slave-updates` enabled. Then it will write updates that it receives from M to its own binary log. When S2 changes from M to S1 as its master, it may receive updates from S1 that it has already received from M

Make sure that all slaves have processed any statements in their relay log. On each slave, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all slaves, they can be reconfigured to the new setup. On the slave S1 being promoted to become the master, issue `STOP SLAVE` and `RESET MASTER`.

On the other slaves S2 and S3, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='S1'` (where 'S1' represents the real hostname of S1). To `CHANGE MASTER`, add all information about how to connect to S1 from S2 or S3 (*user, password, port*). In `CHANGE MASTER`, there is no need to specify the name of S1's binary log or binary log position to read from: We know it is the first binary log and position 4, which are the defaults for `CHANGE MASTER`. Finally, use `START SLAVE` on S2 and S3.

Then instruct all WC to direct their statements to S1. From that point on, all updates statements sent by WC to S1 are written to the binary log of S1, which then contains every update statement sent to S1 since M died.

The result is this configuration:



When M is up again, you must issue on it the same `CHANGE MASTER` as that issued on S2 and S3, so that M becomes a slave of S1 and picks up all the WC writes that it missed while it was down. To make M a master again (because it is the most powerful machine, for example), use the preceding procedure as if S1 was unavailable and M was to be the new master. During this procedure, do not forget to run `RESET MASTER` on M before making S1, S2, and S3 slaves of M. Otherwise, they may pick up old WC writes from before the point at which M became unavailable.

Note that there is no synchronization between the different slaves to a master. Some slaves might be ahead of others. This means that the concept outlined in the previous example might not work. In practice, however, the relay logs of different slaves will most likely not be far behind the master, so it would work, anyway (but there is no guarantee).

Q: How do I prevent `GRANT` and `REVOKE` statements from replicating to slave machines?

A: Start the server with the `--replicate-wild-ignore-table=mysql.%` option.

Q: Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on Mac OS X and Windows)?

A: Yes.

Q: Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

A: Yes.

6.11. Troubleshooting Replication

If you have followed the instructions, and your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the master has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. If logging is enabled, `Position` is non-zero. If binary logging is not enabled, verify that you are running the master with the `--log-bin` and `--server-id` options.
- Verify that the slave is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both Yes. If not, verify the options that were used when starting the slave server. For example, `--skip-slave-start` prevents the slave threads from starting until you issue a `START SLAVE` statement.
- If the slave is running, check whether it established a connection to the master. Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see what they display. See [Section 6.3, “Replication Implementation Details”](#). If the I/O thread state says `Connecting to master`, verify the privileges for the replication user on the master, the master hostname, your DNS setup, whether the master is actually running, and whether it is reachable from the slave.
- If the slave was running previously but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modified the data on the slave outside of the slave thread. If the slave stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in [Section 6.7, “Replication Features and Known Problems”](#). If it is a bug, see [Section 6.12, “How to Report Replication Bugs or Problems”](#), for instructions on how to report it.
- If a statement that succeeded on the master refuses to run on the slave, try

the following procedure if it is not feasible to do a full database resynchronization by deleting the slave's databases and copying a new snapshot from the master:

1. Determine whether the affected table on the slave is different from the master table. Try to understand how this happened. Then make the slave's table identical to the master's and run `START SLAVE`.
2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.
3. If you decide that you can skip the next statement from the master, issue the following statements:

```
mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N;  
mysql> START SLAVE;
```

The value of *N* should be 1 if the next statement from the master does not use `AUTO_INCREMENT` or `LAST_INSERT_ID()`. Otherwise, the value should be 2. The reason for using a value of 2 for statements that use `AUTO_INCREMENT` or `LAST_INSERT_ID()` is that they take two events in the binary log of the master.

4. If you are sure that the slave started out perfectly synchronized with the master, and that no one has updated the tables involved outside of the slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

6.12. How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#). If you have a “phantom” problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the slave thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*
2. Run the slave with the `--log-slave-updates` and `--log-bin` options. These options cause the slave to log the updates that it receives from the master into its own binary logs.
3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:
 - All binary logs from the master
 - All binary logs from the slave
 - The output of `SHOW MASTER STATUS` from the master at the time you discovered the problem
 - The output of `SHOW SLAVE STATUS` from the slave at the time you discovered the problem
 - Error logs from the master and the slave

4. Use **mysqlbinlog** to examine the binary logs. The following should be helpful to find the problem statement. *log_pos* and *log_file* are the Master_Log_File and Read_Master_Log_Pos values from SHOW SLAVE STATUS.

```
shell> mysqlbinlog -j log_pos log_file | head
```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at [Section 1.8, “How to Report Bugs or Problems”](#).

6.13. Auto-Increment in Multiple-Master Replication

When multiple servers are configured as replication masters, special steps must be taken to prevent key collisions when using `AUTO_INCREMENT` columns, otherwise multiple masters may attempt to use the same `AUTO_INCREMENT` value when inserting rows.

The `auto_increment_increment` and `auto_increment_offset` system variables help to accommodate multiple-master replication with `AUTO_INCREMENT` columns. Each of these variables has a default and minimum value of 1, and a maximum value of 65,535. They were introduced in MySQL 5.0.2.

These two variables effect `AUTO_INCREMENT` column behavior as follows:

- `auto_increment_increment` controls the increment between successive `AUTO_INCREMENT` values.
- `auto_increment_offset` determines the starting point for `AUTO_INCREMENT` column values.

By choosing non-conflicting values for these variables on different masters, servers in a multiple-master configuration will not use conflicting `AUTO_INCREMENT` values when inserting new rows into the same table. To set up N master servers, set the variables like this:

- Set `auto_increment_increment` to N on each master.
- Set each of the N masters to have a different `auto_increment_offset`, using the values 1, 2, ..., N .

For additional information about `auto_increment_increment` and `auto_increment_offset`, see [Section 5.2.2, “Server System Variables”](#).

Chapter 7. Optimization

Table of Contents

[7.1. Optimization Overview](#)

[7.1.1. MySQL Design Limitations and Tradeoffs](#)

[7.1.2. Designing Applications for Portability](#)

[7.1.3. What We Have Used MySQL For](#)

[7.1.4. The MySQL Benchmark Suite](#)

[7.1.5. Using Your Own Benchmarks](#)

[7.2. Optimizing SELECT and Other Statements](#)

[7.2.1. Optimizing Queries with EXPLAIN](#)

[7.2.2. Estimating Query Performance](#)

[7.2.3. Speed of SELECT Queries](#)

[7.2.4. WHERE Clause Optimization](#)

[7.2.5. Range Optimization](#)

[7.2.6. Index Merge Optimization](#)

[7.2.7. IS NULL Optimization](#)

[7.2.8. DISTINCT Optimization](#)

[7.2.9. LEFT JOIN and RIGHT JOIN Optimization](#)

[7.2.10. Nested Join Optimization](#)

[7.2.11. Outer Join Simplification](#)

[7.2.12. ORDER BY Optimization](#)

[7.2.13. GROUP BY Optimization](#)

[7.2.14. LIMIT Optimization](#)

[7.2.15. How to Avoid Table Scans](#)

[7.2.16. Speed of INSERT Statements](#)

[7.2.17. Speed of UPDATE Statements](#)

[7.2.18. Speed of DELETE Statements](#)

[7.2.19. Other Optimization Tips](#)

[7.3. Locking Issues](#)

[7.3.1. Locking Methods](#)

[7.3.2. Table Locking Issues](#)

[7.3.3. Concurrent Inserts](#)

[7.4. Optimizing Database Structure](#)

[7.4.1. Design Choices](#)

[7.4.2. Make Your Data as Small as Possible](#)

- [7.4.3. Column Indexes](#)
- [7.4.4. Multiple-Column Indexes](#)
- [7.4.5. How MySQL Uses Indexes](#)
- [7.4.6. The MyISAM Key Cache](#)
- [7.4.7. MyISAM Index Statistics Collection](#)
- [7.4.8. How MySQL Opens and Closes Tables](#)
- [7.4.9. Drawbacks to Creating Many Tables in the Same Database](#)
- [7.5. Optimizing the MySQL Server](#)
 - [7.5.1. System Factors and Startup Parameter Tuning](#)
 - [7.5.2. Tuning Server Parameters](#)
 - [7.5.3. Controlling Query Optimizer Performance](#)
 - [7.5.4. How Compiling and Linking Affects the Speed of MySQL](#)
 - [7.5.5. How MySQL Uses Memory](#)
 - [7.5.6. How MySQL Uses DNS](#)
- [7.6. Disk Issues](#)
 - [7.6.1. Using Symbolic Links](#)

Optimization is a complex task because ultimately it requires understanding of the entire system to be optimized. Although it may be possible to perform some local optimizations with little knowledge of your system or application, the more optimal you want your system to become, the more you must know about it.

This chapter tries to explain and give some examples of different ways to optimize MySQL. Remember, however, that there are always additional ways to make the system even faster, although they may require increasing effort to achieve.

7.1. Optimization Overview

The most important factor in making a system fast is its basic design. You must also know what kinds of processing your system is doing, and what its bottlenecks are. In most cases, system bottlenecks arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When we have the data in main memory, we need to process it to get our result. Having small tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.
- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

7.1.1. MySQL Design Limitations and Tradeoffs

When using the MyISAM storage engine, MySQL uses extremely fast table locking that allows multiple readers or a single writer. The biggest problem with this storage engine occurs when you have a steady stream of mixed updates and slow selects on a single table. If this is a problem for certain tables, you can use another storage engine for them. See [Chapter 14, Storage Engines and Table Types](#).

MySQL can work with both transactional and non-transactional tables. To make it easier to work smoothly with non-transactional tables (which cannot roll back if something goes wrong), MySQL has the following rules. Note that these rules

apply *only* when not running in strict SQL mode or if you use the IGNORE specifier for INSERT or UPDATE.

- All columns have default values.
- If you insert an inappropriate or out-of-range value into a column, MySQL sets the column to the “best possible value” instead of reporting an error. For numerical values, this is 0, the smallest possible value or the largest possible value. For strings, this is either the empty string or as much of the string as can be stored in the column.
- All calculated expressions return a value that can be used instead of signaling an error condition. For example, 1/0 returns NULL.

To change the preceding behaviors, you can enable stricter data handling by setting the server SQL mode appropriately. For more information about data handling, see [Section 1.9.6, “How MySQL Deals with Constraints”](#), [Section 5.2.5, “The Server SQL Mode”](#), and [Section 13.2.4, “INSERT Syntax”](#).

7.1.2. Designing Applications for Portability

Because all SQL servers implement different parts of standard SQL, it takes work to write portable database applications. It is very easy to achieve portability for very simple selects and inserts, but becomes more difficult the more capabilities you require. If you want an application that is fast with many database systems, it becomes even more difficult.

All database systems have some weak points. That is, they have different design compromises that lead to different behavior.

To make a complex application portable, you need to determine which SQL servers it must work with, and then determine what features those servers support. You can use the MySQL **crash-me** program to find functions, types, and limits that you can use with a selection of database servers. **crash-me** does not check for every possible feature, but it is still reasonably comprehensive, performing about 450 tests. An example of the type of information **crash-me** can provide is that you should not use column names that are longer than 18 characters if you want to be able to use Informix or DB2.

The **crash-me** program and the MySQL benchmarks are all very database independent. By taking a look at how they are written, you can get a feeling for what you must do to make your own applications database independent. The programs can be found in the `sql-bench` directory of MySQL source distributions. They are written in Perl and use the DBI database interface. Use of DBI in itself solves part of the portability problem because it provides database-independent access methods. See [Section 7.1.4, “The MySQL Benchmark Suite”](#).

If you strive for database independence, you need to get a good feeling for each SQL server's bottlenecks. For example, MySQL is very fast in retrieving and updating rows for MyISAM tables, but has a problem in mixing slow readers and writers on the same table. Oracle, on the other hand, has a big problem when you try to access rows that you have recently updated (until they are flushed to disk). Transactional database systems in general are not very good at generating summary tables from log tables, because in this case row locking is almost useless.

To make your application *really* database independent, you should define an easily extendable interface through which you manipulate your data. For example, C++ is available on most systems, so it makes sense to use a C++ class-based interface to the databases.

If you use some feature that is specific to a given database system (such as the REPLACE statement, which is specific to MySQL), you should implement the same feature for other SQL servers by coding an alternative method. Although the alternative might be slower, it enables the other servers to perform the same tasks.

With MySQL, you can use the `/*! */` syntax to add MySQL-specific keywords to a statement. The code inside `/* */` is treated as a comment (and ignored) by most other SQL servers. For information about writing comments, see [Section 9.4, “Comment Syntax”](#).

If high performance is more important than exactness, as for some Web applications, it is possible to create an application layer that caches all results to give you even higher performance. By letting old results expire after a while, you can keep the cache reasonably fresh. This provides a method to handle high load spikes, in which case you can dynamically increase the cache size and set

the expiration timeout higher until things get back to normal.

In this case, the table creation information should contain information about the initial cache size and how often the table should normally be refreshed.

An attractive alternative to implementing an application cache is to use the MySQL query cache. By enabling the query cache, the server handles the details of determining whether a query result can be reused. This simplifies your application. See [Section 5.14, “The MySQL Query Cache”](#).

7.1.3. What We Have Used MySQL For

This section describes an early application for MySQL.

During MySQL initial development, the features of MySQL were made to fit our largest customer, which handled data warehousing for a couple of the largest retailers in Sweden.

From all stores, we got weekly summaries of all bonus card transactions, and were expected to provide useful information for the store owners to help them find how their advertising campaigns were affecting their own customers.

The volume of data was quite huge (about seven million summary transactions per month), and we had data for 4–10 years that we needed to present to the users. We got weekly requests from our customers, who wanted instant access to new reports from this data.

We solved this problem by storing all information per month in compressed “transaction tables.” We had a set of simple macros that generated summary tables grouped by different criteria (product group, customer id, store, and so on) from the tables in which the transactions were stored. The reports were Web pages that were dynamically generated by a small Perl script. This script parsed a Web page, executed the SQL statements in it, and inserted the results. We would have used PHP or **mod_perl** instead, but they were not available at the time.

For graphical data, we wrote a simple tool in C that could process SQL query results and produce GIF images based on those results. This tool also was dynamically executed from the Perl script that parses the Web pages.

In most cases, a new report could be created simply by copying an existing script and modifying the SQL query that it used. In some cases, we needed to add more columns to an existing summary table or generate a new one. This also was quite simple because we kept all transaction-storage tables on disk. (This amounted to about 50GB of transaction tables and 200GB of other customer data.)

We also let our customers access the summary tables directly with ODBC so that the advanced users could experiment with the data themselves.

This system worked well and we had no problems handling the data with quite modest Sun Ultra SPARCstation hardware (2×200MHz). Eventually the system was migrated to Linux.

7.1.4. The MySQL Benchmark Suite

This benchmark suite is meant to tell any user what operations a given SQL implementation performs well or poorly. You can get a good idea for how the benchmarks work by looking at the code and results in the `sql-bench` directory in any MySQL source distribution.

Note that this benchmark is single-threaded, so it measures the minimum time for the operations performed. We plan to add multi-threaded tests to the benchmark suite in the future.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions. You can either download a released distribution from <http://dev.mysql.com/downloads/>, or use the current development source tree. (See [Section 2.9.3, “Installing from the Development Source Tree”](#).)
- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You also need the server-specific DBD drivers for each of the servers you want to test. For example, to test MySQL, PostgreSQL, and DB2, you must have the `DBD::mysql`, `DBD::Pg`, and `DBD::DB2` modules installed. See [Section 2.14, “Perl Installation Notes”](#).

After you obtain a MySQL source distribution, you can find the benchmark suite

located in its `sql-bench` directory. To run the benchmark tests, build MySQL, and then change location into the `sql-bench` directory and execute the `run-all-tests` script:

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

`server_name` should be the name of one of the supported servers. To get a list of all options and supported servers, invoke this command:

```
shell> perl run-all-tests --help
```

The **crash-me** script also is located in the `sql-bench` directory. **crash-me** tries to determine what features a database system supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What data types are supported
- How many indexes are supported
- What functions are supported
- How big a query can be
- How big a `VARCHAR` column can be

You can find the results from **crash-me** for many different database servers at <http://dev.mysql.com/tech-resources/crash-me.php>. For more information about benchmark results, visit <http://dev.mysql.com/tech-resources/benchmarks/>.

7.1.5. Using Your Own Benchmarks

You should definitely benchmark your application and database to find out where the bottlenecks are. After fixing one bottleneck (or by replacing it with a “dummy” module), you can proceed to identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck and decide how to solve it if someday you really need the extra performance.

For examples of portable benchmark programs, look at those in the MySQL benchmark suite. See [Section 7.1.4, “The MySQL Benchmark Suite”](#). You can

take any program from this suite and modify it for your own needs. By doing this, you can try different solutions to your problem and test which really is fastest for you.

Another free benchmark suite is the Open Source Database Benchmark, available at <http://osdb.sourceforge.net/>.

It is very common for a problem to occur only when the system is very heavily loaded. We have had many customers who contact us when they have a (tested) system in production and have encountered load problems. In most cases, performance problems turn out to be due to issues of basic database design (for example, table scans are not good under high load) or problems with the operating system or libraries. Most of the time, these problems would be much easier to fix if the systems were not already in production.

To avoid problems like this, you should put some effort into benchmarking your whole application under the worst possible load. You can use Super Smack, available at <http://jeremy.zawodny.com/mysql/super-smack/>. As suggested by its name, it can bring a system to its knees, so make sure to use it only on your development systems.

7.2. Optimizing SELECT and Other Statements

First, one factor affects all statements: The more complex your permissions setup, the more overhead you have. Using simpler permissions when you issue GRANT statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you do not grant any table-level or column-level privileges, the server need not ever check the contents of the `tables_priv` and `columns_priv` tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high statement-processing load, it may be worth the time to use a simplified grant structure to reduce permission-checking overhead.

If your problem is with a specific MySQL expression or function, you can perform a timing test by invoking the `BENCHMARK()` function using the **mysql** client program. Its syntax is `BENCHMARK(loop_count,expression)`. The return value is always zero, but **mysql** prints a line displaying approximately how long the statement took to execute. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+-----+
| BENCHMARK(1000000,1+1) |
+-----+
|                          0 |
+-----+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

All MySQL functions should be highly optimized, but there may be some exceptions. `BENCHMARK()` is an excellent tool for finding out if some function is a problem for your queries.

7.2.1. Optimizing Queries with EXPLAIN

```
EXPLAIN tbl_name
```

Or:

```
EXPLAIN [EXTENDED] SELECT select_options
```

The EXPLAIN statement can be used either as a synonym for DESCRIBE or as a way to obtain information about how MySQL executes a SELECT statement:

- EXPLAIN tbl_name is synonymous with DESCRIBE tbl_name or SHOW COLUMNS FROM tbl_name.
- When you precede a SELECT statement with the keyword EXPLAIN, MySQL displays information from the optimizer about the query execution plan. That is, MySQL explains how it would process the SELECT, including information about how tables are joined and in which order.

This section describes the second use of EXPLAIN for obtaining query execution plan information. For a description of the DESCRIBE and SHOW COLUMNS statements, see [Section 13.3.1, “DESCRIBE Syntax”](#), and [Section 13.5.4.3, “SHOW COLUMNS Syntax”](#).

With the help of EXPLAIN, you can see where you should add indexes to tables to get a faster SELECT that uses indexes to find rows. You can also use EXPLAIN to check whether the optimizer joins the tables in an optimal order. To force the optimizer to use a join order corresponding to the order in which the tables are named in the SELECT statement, begin the statement with SELECT STRAIGHT_JOIN rather than just SELECT.

If you have a problem with indexes not being used when you believe that they should be, you should run ANALYZE TABLE to update table statistics such as cardinality of keys, that can affect the choices the optimizer makes. See [Section 13.5.2.1, “ANALYZE TABLE Syntax”](#).

EXPLAIN returns a row of information for each table used in the SELECT statement. The tables are listed in the output in the order that MySQL would read them while processing the query. MySQL resolves all joins using a *single-sweep multi-join* method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

When the EXTENDED keyword is used, EXPLAIN produces extra information that can be viewed by issuing a SHOW WARNINGS statement following the EXPLAIN statement. This information displays how the optimizer qualifies table and column names in the SELECT statement, what the SELECT looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process.

Each output row from EXPLAIN provides information about one table, and each row contains the following columns:

- id

The SELECT identifier. This is the sequential number of the SELECT within the query.

- select_type

The type of SELECT, which can be any of those shown in the following table:

SIMPLE	Simple SELECT (not using UNION or subqueries)
PRIMARY	Outermost SELECT
UNION	Second or later SELECT statement in a UNION
DEPENDENT UNION	Second or later SELECT statement in a UNION, dependent on outer query
UNION RESULT	Result of a UNION.
SUBQUERY	First SELECT in subquery
DEPENDENT SUBQUERY	First SELECT in subquery, dependent on outer query
DERIVED	Derived table SELECT (subquery in FROM clause)

DEPENDENT typically signifies the use of a correlated subquery. See [Section 13.2.8.7, “Correlated Subqueries”](#).

- table

The table to which the row of output refers.

- type

The join type. The different join types are listed here, ordered from the best type to the worst:

- system

The table has only one row (= system table). This is a special case of the const join type.

- const

The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. const tables are very fast because they are read only once.

const is used when you compare all parts of a PRIMARY KEY or UNIQUE index to constant values. In the following queries, *tbl_name* can be used as a const table:

```
SELECT * FROM tbl_name WHERE primary_key=1;
```

```
SELECT * FROM tbl_name  
WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- eq_ref

One row is read from this table for each combination of rows from the previous tables. Other than the system and const types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a PRIMARY KEY or UNIQUE index.

eq_ref can be used for indexed columns that are compared using the = operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table. In the following examples, MySQL can use an eq_ref join to process *ref_table*:

```
SELECT * FROM ref_table,other_table  
WHERE ref_table.key_column=other_table.column;
```

```
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

- `ref`

All rows with matching index values are read from this table for each combination of rows from the previous tables. `ref` is used if the join uses only a leftmost prefix of the key or if the key is not a PRIMARY KEY or UNIQUE index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

`ref` can be used for indexed columns that are compared using the = or <=> operator. In the following examples, MySQL can use a `ref` join to process *ref_table*:

```
SELECT * FROM ref_table WHERE key_column=expr;
```

```
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;
```

```
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

- `ref_or_null`

This join type is like `ref`, but with the addition that MySQL does an extra search for rows that contain NULL values. This join type optimization is used most often in resolving subqueries. In the following examples, MySQL can use a `ref_or_null` join to process *ref_table*:

```
SELECT * FROM ref_table
  WHERE key_column=expr OR key_column IS NULL;
```

See [Section 7.2.7, “IS NULL Optimization”](#).

- `index_merge`

This join type indicates that the Index Merge optimization is used. In this case, the key column in the output row contains a list of indexes

used, and `key_len` contains a list of the longest key parts for the indexes used. For more information, see [Section 7.2.6, “Index Merge Optimization”](#).

- `unique_subquery`

This type replaces `ref` for some IN subqueries of the following form:

```
value IN (SELECT primary_key FROM single_table WHERE some_exp)
```

`unique_subquery` is just an index lookup function that replaces the subquery completely for better efficiency.

- `index_subquery`

This join type is similar to `unique_subquery`. It replaces IN subqueries, but it works for non-unique indexes in subqueries of the following form:

```
value IN (SELECT key_column FROM single_table WHERE some_exp)
```

- `range`

Only rows that are in a given range are retrieved, using an index to select the rows. The key column in the output row indicates which index is used. The `key_len` contains the longest key part that was used. The `ref` column is NULL for this type.

`range` can be used when a key column is compared to a constant using any of the `=`, `<>`, `>`, `>=`, `<`, `<=`, `IS NULL`, `<=>`, `BETWEEN`, or `IN` operators:

```
SELECT * FROM tbl_name  
  WHERE key_column = 10;
```

```
SELECT * FROM tbl_name  
  WHERE key_column BETWEEN 10 and 20;
```

```
SELECT * FROM tbl_name  
  WHERE key_column IN (10,20,30);
```

```
SELECT * FROM tbl_name  
  WHERE key_part1= 10 AND key_part2 IN (10,20,30);
```

- index

This join type is the same as ALL, except that only the index tree is scanned. This usually is faster than ALL because the index file usually is smaller than the data file.

MySQL can use this join type when the query uses only columns that are part of a single index.

- ALL

A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked const, and usually *very* bad in all other cases. Normally, you can avoid ALL by adding indexes that allow row retrieval from the table based on constant values or column values from earlier tables.

- possible_keys

The possible_keys column indicates which indexes MySQL can choose from use to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from EXPLAIN. That means that some of the keys in possible_keys might not be usable in practice with the generated table order.

If this column is NULL, there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the WHERE clause to check whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with EXPLAIN again. See [Section 13.1.2, “ALTER TABLE Syntax”](#).

To see what indexes a table has, use SHOW INDEX FROM tbl_name.

- key

The key column indicates the key (index) that MySQL actually decided to use. The key is NULL if no index was chosen. To force MySQL to use or ignore an index listed in the possible_keys column, use FORCE INDEX, USE INDEX, or IGNORE INDEX in your query. See [Section 13.2.7, “SELECT Syntax”](#).

For MyISAM and BDB tables, running `ANALYZE TABLE` helps the optimizer choose better indexes. For MyISAM tables, `myisamchk --analyze` does the same. See [Section 13.5.2.1, “ANALYZE TABLE Syntax”](#), and [Section 5.10.4, “Table Maintenance and Crash Recovery”](#).

- `key_len`

The `key_len` column indicates the length of the key that MySQL decided to use. The length is `NULL` if the key column says `NULL`. Note that the value of `key_len` enables you to determine how many parts of a multiple-part key MySQL actually uses.

- `ref`

The `ref` column shows which columns or constants are compared to the index named in the `key` column to select rows from the table.

- `rows`

The `rows` column indicates the number of rows MySQL believes it must examine to execute the query.

- `Extra`

This column contains additional information about how MySQL resolves the query. Here is an explanation of the values that can appear in this column:

- `Distinct`

MySQL is looking for distinct values, so it stops searching for more rows for the current row combination after it has found the first matching row.

- `Not exists`

MySQL was able to do a `LEFT JOIN` optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the `LEFT JOIN` criteria. Here is an example of the type of query that can be optimized this

way:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
WHERE t2.id IS NULL;
```

Assume that `t2.id` is defined as `NOT NULL`. In this case, MySQL scans `t1` and looks up the rows in `t2` using the values of `t1.id`. If MySQL finds a matching row in `t2`, it knows that `t2.id` can never be `NULL`, and does not scan through the rest of the rows in `t2` that have the same `id` value. In other words, for each row in `t1`, MySQL needs to do only a single lookup in `t2`, regardless of how many rows actually match in `t2`.

- range checked for each record (index map: N)

MySQL found no good index to use, but found that some of indexes might be used after column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a range or `index_merge` access method to retrieve rows. This is not very fast, but is faster than performing a join with no index at all. The applicability criteria are as described in [Section 7.2.5, “Range Optimization”](#), and [Section 7.2.6, “Index Merge Optimization”](#), with the exception that all column values for the preceding table are known and considered to be constants.

- Using `filesort`

MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the `WHERE` clause. The keys then are sorted and the rows are retrieved in sorted order. See [Section 7.2.12, “ORDER BY Optimization”](#).

- Using `index`

The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

- Using temporary

To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains `GROUP BY` and `ORDER BY` clauses that list columns differently.

- Using where

A `WHERE` clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the `Extra` value is not `Using where` and the table join type is `ALL` or `index`.

If you want to make your queries as fast as possible, you should look out for `Extra` values of `Using filesort` and `Using temporary`.

- Using `sort_union(...)`, `Using union(...)`, `Using intersect(...)`

These indicate how index scans are merged for the `index_merge` join type. See [Section 7.2.6, “Index Merge Optimization”](#), for more information.

- Using `index` for group-by

Similar to the `Using index` way of accessing a table, `Using index for group-by` indicates that MySQL found an index that can be used to retrieve all columns of a `GROUP BY` or `DISTINCT` query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see [Section 7.2.13, “GROUP BY Optimization”](#).

- Using `where` with pushed condition

This item applies to `NDB Cluster` tables *only*. It means that MySQL Cluster is using *condition pushdown* to improve the efficiency of a direct comparison (`=`) between a non-indexed column and a constant. In such cases, the condition is “pushed down” to the cluster’s data nodes where it is evaluated in all partitions simultaneously. This eliminates the need to send non-matching rows over the network, and

can speed up such queries by a factor of 5 to 10 times over cases where condition pushdown could be but is not used.

Suppose that you have a Cluster table defined as follows:

```
CREATE TABLE t1 (  
    a INT,  
    b INT,  
    KEY(a)  
) ENGINE=NDBCLUSTER;
```

In this case, condition pushdown can be used with a query such as this one:

```
SELECT a,b FROM t1 WHERE b = 10;
```

This can be seen in the output of EXPLAIN SELECT, as shown here:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G  
***** 1. row *****  
      id: 1  
    select_type: SIMPLE  
      table: t1  
       type: ALL  
possible_keys: NULL  
         key: NULL  
    key_len: NULL  
         ref: NULL  
       rows: 10  
   Extra: Using where with pushed condition
```

Condition pushdown *cannot* be used with either of these two queries:

```
SELECT a,b FROM t1 WHERE a = 10;  
SELECT a,b FROM t1 WHERE b + 1 = 10;
```

With regard to the first of these two queries, condition pushdown is not applicable because an index exists on column a. In the case of the second query, a condition pushdown cannot be employed because the comparison involving the non-indexed column b is an indirect one. (However, it would apply if you were to reduce $b + 1 = 10$ to $b = 9$ in the WHERE clause.)

However, a condition pushdown may also be employed when an

indexed column column is compared with a constant using a > or < operator:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE a<2\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: t1
      type: range
possible_keys: a
      key: a
     key_len: 5
      ref: NULL
      rows: 2
  Extra: Using where with pushed condition
```

With regard to condition pushdown, keep in mind that:

- Condition pushdown is relevant to MySQL Cluster *only*, and does not occur when executing queries against tables using any other storage engine.
- Condition pushdown capability is not used by default. To enable it, you can start **mysqld** with the `--engine-condition-pushdown` option, or execute the following statement:

```
SET engine_condition_pushdown=On;
```

Note: Condition pushdown is not supported for columns of any of the BLOB or TEXT types.

Condition pushdown, Using where with pushed condition, and `engine_condition_pushdown` were all introduced in MySQL 5.0 Cluster.

You can get a good indication of how good a join is by taking the product of the values in the `rows` column of the EXPLAIN output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the `max_join_size` system variable, this row product also is used to determine which multiple-table SELECT statements to execute and which to abort. See [Section 7.5.2, “Tuning Server Parameters”](#).

The following example shows how a multiple-table join can be optimized

progressively based on the information provided by EXPLAIN.

Suppose that you have the SELECT statement shown here and that you plan to examine it using EXPLAIN:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
              tt.ProjectReference, tt.EstimatedShipDate,
              tt.ActualShipDate, tt.ClientID,
              tt.ServiceCodes, tt.RepetitiveID,
              tt.CurrentProcess, tt.CurrentDPPerson,
              tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
              et_1.COUNTRY, do.CUSTNAME
FROM tt, et, et AS et_1, do
WHERE tt.SubmitTime IS NULL
      AND tt.ActualPC = et.EMPLOYID
      AND tt.AssignedPC = et_1.EMPLOYID
      AND tt.ClientID = do.CUSTNMBR;
```

For this example, make the following assumptions:

- The columns being compared have been declared as follows:

Table	Column	Data Type
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- The tables have the following indexes:

Table	Index
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The tt.ActualPC values are not evenly distributed.

Initially, before any optimizations have been performed, the EXPLAIN statement

produces the following information:

```
table type possible_keys key key_len ref rows Extra
et ALL PRIMARY NULL NULL NULL 74
do ALL PRIMARY NULL NULL NULL 2135
et_1 ALL PRIMARY NULL NULL NULL 74
tt ALL AssignedPC, NULL NULL NULL 3872
      ClientID,
      ActualPC
      range checked for each record (key map: 35)
```

Because type is ALL for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is $74 \times 2135 \times 74 \times 3872 = 45,268,558,720$ rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, VARCHAR and CHAR are considered the same if they are declared as the same size. tt.ActualPC is declared as CHAR(10) and et.EMPLOYID is CHAR(15), so there is a length mismatch.

To fix this disparity between column lengths, use ALTER TABLE to lengthen ActualPC from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now tt.ActualPC and et.EMPLOYID are both VARCHAR(15). Executing the EXPLAIN statement again produces this result:

```
table type possible_keys key key_len ref rows Extra
tt ALL AssignedPC, NULL NULL NULL 3872 Using
      ClientID, where
      ActualPC
do ALL PRIMARY NULL NULL NULL 2135
      range checked for each record (key map: 1)
et_1 ALL PRIMARY NULL NULL NULL 74
      range checked for each record (key map: 1)
et eq_ref PRIMARY PRIMARY 15 tt.ActualPC 1
```

This is not perfect, but is much better: The product of the rows values is less by a factor of 74. This version executes in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the `tt.AssignedPC = et_1.EMPLOYID` and `tt.ClientID = do.CUSTNMBR` comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
->          MODIFY ClientID VARCHAR(15);
```

After that modification, `EXPLAIN` produces the output shown here:

table	type	possible_keys	key	key_len	ref	rows	Extra
et	ALL	PRIMARY	NULL	NULL	NULL	74	
tt	ref	AssignedPC, ClientID, ActualPC	ActualPC	15	et.EMPLOYID	52	Using where
et_1	eq_ref	PRIMARY	PRIMARY	15	tt.AssignedPC	1	
do	eq_ref	PRIMARY	PRIMARY	15	tt.ClientID	1	

At this point, the query is optimized almost as well as possible. The remaining problem is that, by default, MySQL assumes that values in the `tt.ActualPC` column are evenly distributed, and that is not the case for the `tt` table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

With the additional index information, the join is perfect and `EXPLAIN` produces this result:

table	type	possible_keys	key	key_len	ref	rows	Extra
tt	ALL	AssignedPC, ClientID, ActualPC	NULL	NULL	NULL	3872	Using where
et	eq_ref	PRIMARY	PRIMARY	15	tt.ActualPC	1	
et_1	eq_ref	PRIMARY	PRIMARY	15	tt.AssignedPC	1	
do	eq_ref	PRIMARY	PRIMARY	15	tt.ClientID	1	

Note that the `rows` column in the output from `EXPLAIN` is an educated guess from the MySQL join optimizer. You should check whether the numbers are even close to the truth by comparing the `rows` product with the actual number of rows that the query returns. If the numbers are quite different, you might get better performance by using `STRAIGHT_JOIN` in your `SELECT` statement and trying to list the tables in a different order in the `FROM` clause.

7.2.2. Estimating Query Performance

In most cases, you can estimate query performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: $\log(\text{row_count}) / \log(\text{index_block_length} / 3 \times 2 / (\text{index_length} + \text{data_pointer_length})) + 1$.

In MySQL, an index block is usually 1,024 bytes and the data pointer is usually four bytes. For a 500,000-row table with an index length of three bytes (the size of `MEDIUMINT`), the formula indicates $\log(500,000) / \log(1024/3 \times 2 / (3+4)) + 1 = 4$ seeks.

This index would require storage of about $500,000 \times 7 \times 3/2 = 5.2\text{MB}$ (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and so need only one or two calls to read data to find the row.

For writes, however, you need four seek requests to find where to place a new index value and normally two seeks to update the index and write the row.

Note that the preceding discussion does not mean that your application performance slowly degenerates by $\log N$. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are bound only by disk seeks (which increase by $\log N$). To avoid this, increase the key cache size as the data grows. For MyISAM tables, the key cache size is controlled by the `key_buffer_size` system variable. See [Section 7.5.2, “Tuning Server Parameters”](#).

7.2.3. Speed of SELECT Queries

In general, when you want to make a slow `SELECT ... WHERE` query faster, the first thing to check is whether you can add an index. All references between different tables should usually be done with indexes. You can use the `EXPLAIN` statement to determine which indexes are used for a `SELECT`. See [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#), and [Section 7.4.5, “How MySQL Uses Indexes”](#).

Some general tips for speeding up queries on MyISAM tables:

- To help MySQL better optimize queries, use `ANALYZE TABLE` or run **`myisamchk --analyze`** on a table after it has been loaded with data. This updates a value for each index part that indicates the average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a non-constant expression. You can check the result from the table analysis by using `SHOW INDEX FROM tbl_name` and examining the Cardinality value. **`myisamchk --description --verbose`** shows index distribution information.
- To sort an index and data according to an index, use **`myisamchk --sort-index --sort-records=1`** (assuming that you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all rows in order according to the index. The first time you sort a large table this way, it may take a long time.

7.2.4. WHERE Clause Optimization

This section discusses optimizations that can be made for processing `WHERE` clauses. The examples use `SELECT` statements, but the same optimizations apply for `WHERE` clauses in `DELETE` and `UPDATE` statements.

Work on the MySQL optimizer is ongoing, so this section is incomplete. MySQL performs a great many optimizations, not all of which are documented here.

Some of the optimizations performed by MySQL follow:

- Removal of unnecessary parentheses:

```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
(a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal (needed because of constant folding):

```
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- Constant expressions used by indexes are evaluated only once.
- `COUNT(*)` on a single table without a `WHERE` is retrieved directly from the table information for `MyISAM` and `MEMORY` tables. This is also done for any `NOT NULL` expression when used with only one table.
- Early detection of invalid constant expressions. MySQL quickly detects that some `SELECT` statements are impossible and returns no rows.
- `HAVING` is merged with `WHERE` if you do not use `GROUP BY` or aggregate functions (`COUNT()`, `MIN()`, and so on).
- For each table in a join, a simpler `WHERE` is constructed to get a fast `WHERE` evaluation for the table and also to skip rows as soon as possible.
- All constant tables are read first before any other tables in the query. A constant table is any of the following:
 - An empty table or a table with one row.
 - A table that is used with a `WHERE` clause on a `PRIMARY KEY` or a `UNIQUE` index, where all index parts are compared to constant expressions and are defined as `NOT NULL`.

All of the following tables are used as constant tables:

```
SELECT * FROM t WHERE primary_key=1;
SELECT * FROM t1,t2
  WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in `ORDER BY` and `GROUP BY` clauses come from the same table, that table is preferred first when joining.
- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use the `SQL_SMALL_RESULT` option, MySQL uses an in-memory temporary table.

- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table, but a fixed percentage no longer determines the choice between using an index or a scan. The optimizer now is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size.
- In some cases, MySQL can read rows from the index without even consulting the data file. If all columns used from the index are numeric, only the index tree is used to resolve the query.
- Before each row is output, those that do not match the HAVING clause are skipped.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;
```

```
SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;
```

```
SELECT MAX(key_part2) FROM tbl_name
WHERE key_part1=constant;
```

```
SELECT ... FROM tbl_name
ORDER BY key_part1,key_part2,... LIMIT 10;
```

```
SELECT ... FROM tbl_name
ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL resolves the following queries using only the index tree, assuming that the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val1;
```

```
SELECT COUNT(*) FROM tbl_name
WHERE key_part1=val1 AND key_part2=val2;
```

```
SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
ORDER BY key_part1,key_part2,... ;
```

```
SELECT ... FROM tbl_name
ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

7.2.5. Range Optimization

The range access method uses a single index to retrieve a subset of table rows that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. The following sections give a detailed description of how intervals are extracted from the `WHERE` clause.

7.2.5.1. The Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the `WHERE` clause, so we speak of *range conditions* rather than “intervals.”

The definition of a range condition for a single-part index is as follows:

- For both `BTREE` and `HASH` indexes, comparison of a key part with a constant value is a range condition when using the `=`, `<=>`, `IN`, `IS NULL`, or `IS NOT NULL` operators.
- For `BTREE` indexes, comparison of a key part with a constant value is a range condition when using the `>`, `<`, `>=`, `<=`, `BETWEEN`, `!=`, or `<>` operators, or `LIKE 'pattern'` (where `'pattern'` does not start with a wildcard).
- For all types of indexes, multiple range conditions combined with `OR` or `AND` form a range condition.

“Constant value” in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a const or system table from the same join
- The result of an uncorrelated subquery
- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the WHERE clause:

```
SELECT * FROM t1
  WHERE key_col > 1
  AND key_col < 10;
```

```
SELECT * FROM t1
  WHERE key_col = 1
  OR key_col IN (15, 18, 20);
```

```
SELECT * FROM t1
  WHERE key_col LIKE 'ab%'
  OR key_col BETWEEN 'bar' AND 'foo';
```

Note that some non-constant values may be converted to constants during the constant propagation phase.

MySQL tries to extract range conditions from the WHERE clause for each of the possible indexes. During the extraction process, conditions that cannot be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

Consider the following statement, where key1 is an indexed column and nonkey is not indexed:

```
SELECT * FROM t1 WHERE
  (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
  (key1 < 'bar' AND nonkey = 4) OR
  (key1 < 'uux' AND key1 > 'z');
```

The extraction process for key key1 is as follows:

1. Start with original WHERE clause:

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z')
```

2. Remove nonkey = 4 and key1 LIKE '%b' because they cannot be used for a range scan. The correct way to remove them is to replace them with TRUE, so that we do not miss any matching rows when doing the range scan. Having replaced them with TRUE, we get:

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR  
(key1 < 'bar' AND TRUE) OR  
(key1 < 'uux' AND key1 > 'z')
```

3. Collapse conditions that are always true or false:

- (key1 LIKE 'abcde%' OR TRUE) is always true
- (key1 < 'uux' AND key1 > 'z') is always false

Replacing these conditions with constants, we get:

```
(key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
```

Removing unnecessary TRUE and FALSE constants, we obtain:

```
(key1 < 'abc') OR (key1 < 'bar')
```

4. Combining overlapping intervals into one yields the final condition to be used for the range scan:

```
(key1 < 'bar')
```

In general (and as demonstrated by the preceding example), the condition used for a range scan is less restrictive than the WHERE clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full WHERE clause.

The range condition extraction algorithm can handle nested AND/OR constructs of arbitrary depth, and its output does not depend on the order in which conditions appear in WHERE clause.

7.2.5.2. The Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index rows to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as `key1(key_part1, key_part2, key_part3)`, and the following set of key tuples listed in key order:

<i>key_part1</i>	<i>key_part2</i>	<i>key_part3</i>
NULL	1	'abc'
NULL	1	'xyz'
NULL	2	'foo'
1	1	'abc'
1	1	'xyz'
1	2	'abc'
2	1	'aaa'

The condition `key_part1 = 1` defines this interval:

$$(1, -inf, -inf) \leq (key_part1, key_part2, key_part3) < (1, +inf, +inf)$$

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition `key_part3 = 'abc'` does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For HASH indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

```

    key_part1 cmp const1
AND key_part2 cmp const2
AND ...
AND key_partN cmp constN;

```

Here, *const1*, *const2*, ... are constants, *cmp* is one of the =, <=>, or IS NULL comparison operators, and the conditions cover all index parts. (That is, there are *N* conditions, one for each part of an *N*-part index.) For example, the following is a range condition for a three-part HASH index:

```
key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
```

For the definition of what is considered to be a constant, see [Section 7.2.5.1, “The Range Access Method for Single-Part Indexes”](#).

- For a BTREE index, an interval might be usable for conditions combined with AND, where each condition compares a key part with a constant value

using =, <=>, IS NULL, >, <, >=, <=, !=, <>, BETWEEN, or LIKE 'pattern' (where 'pattern' does not start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all rows that match the condition (or two intervals if <> or != is used). For example, for this condition:

```
key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
```

The single interval is:

```
('foo', 10, 10) < (key_part1, key_part2, key_part3) < ('foo', +inf, +j
```

It is possible that the created interval contains more rows than the initial condition. For example, the preceding interval includes the value ('foo', 11, 0), which does not satisfy the original condition.

- If conditions that cover sets of rows contained within intervals are combined with OR, they form a condition that covers a set of rows contained within the union of their intervals. If the conditions are combined with AND, they form a condition that covers a set of rows contained within the intersection of their intervals. For example, for this condition on a two-part index:

```
(key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
```

The intervals are:

```
(1, -inf) < (key_part1, key_part2) < (1, 2)  
(5, -inf) < (key_part1, key_part2)
```

In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The `key_1en` column in the EXPLAIN output indicates the maximum length of the key prefix used.

In some cases, `key_1en` may indicate that a key part was used, but that might be not what you would expect. Suppose that `key_part1` and `key_part2` can be NULL. Then the `key_1en` column displays two key part lengths for the following condition:

```
key_part1 >= 1 AND key_part2 < 2
```

But, in fact, the condition is converted to this:

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

[Section 7.2.5.1, “The Range Access Method for Single-Part Indexes”](#), describes how optimizations are performed to combine or eliminate intervals for range conditions on a single-part index. Analogous steps are performed for range conditions on multiple-part indexes.

7.2.6. Index Merge Optimization

The *Index Merge* method is used to retrieve rows with several range scans and to merge their results into one. The merge can produce unions, intersections, or unions-of-intersections of its underlying scans.

Note: If you have upgraded from a previous version of MySQL, you should be aware that this type of join optimization is first introduced in MySQL 5.0, and represents a significant change in behavior with regard to indexes. (Formerly, MySQL was able to use at most only one index for each referenced table.)

In EXPLAIN output, the Index Merge method appears as `index_merge` in the `type` column. In this case, the `key` column contains a list of indexes used, and `key_len` contains a list of the longest key parts for those indexes.

Examples:

```
SELECT * FROM tbl_name WHERE key_part1 = 10 OR key_part2 = 20;
```

```
SELECT * FROM tbl_name
  WHERE (key_part1 = 10 OR key_part2 = 20) AND non_key_part=30;
```

```
SELECT * FROM t1, t2
  WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
  AND t2.key1=t1.some_col;
```

```
SELECT * FROM t1, t2
  WHERE t1.key1=1
  AND (t2.key1=t1.some_col OR t2.key2=t1.some_col2);
```

The Index Merge method has several access algorithms (seen in the `Extra` field of EXPLAIN output):

- Using `intersect(...)`
- Using `union(...)`
- Using `sort_union(...)`

The following sections describe these methods in greater detail.

Note: The Index Merge optimization algorithm has the following known deficiencies:

- If a range scan is possible on some key, an Index Merge is not considered. For example, consider this query:

```
SELECT * FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

For this query, two plans are possible:

- An Index Merge scan using the `(goodkey1 < 10 OR goodkey2 < 20)` condition.
- A range scan using the `badkey < 30` condition.

However, the optimizer considers only the second plan. If that is not what you want, you can make the optimizer consider Index Merge by using `IGNORE INDEX` or `FORCE INDEX`. The following queries are executed using Index Merge:

```
SELECT * FROM t1 FORCE INDEX(index_for_goodkey1,index_for_goodkey2)
WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

```
SELECT * FROM t1 IGNORE INDEX(index_for_badkey)
WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

- If your query has a complex `WHERE` clause with deep `AND/OR` nesting and MySQL doesn't choose the optimal plan, try distributing terms using the following identity laws:

$$(x \text{ AND } y) \text{ OR } z = (x \text{ OR } z) \text{ AND } (y \text{ OR } z)$$

$$(x \text{ OR } y) \text{ AND } z = (x \text{ AND } z) \text{ OR } (y \text{ AND } z)$$

- Index Merge is not applicable to fulltext indexes. We plan to extend it to

cover these in a future MySQL release.

The choice between different possible variants of the Index Merge access method and other access methods is based on cost estimates of various available options.

7.2.6.1. The Index Merge Intersection Access Algorithm

This access algorithm can be employed when a WHERE clause was converted to several range conditions on different keys combined with AND, and each condition is one of the following:

- In this form, where the index has exactly N parts (that is, all index parts are covered):

key_part1=const1 AND key_part2=const2 ... AND key_partN=constN

- Any range condition over a primary key of an InnoDB or BDB table.

Examples:

```
SELECT * FROM innodb_table WHERE primary_key < 10 AND key_col1=20;
```

```
SELECT * FROM tbl_name  
WHERE (key1_part1=1 AND key1_part2=2) AND key2=2;
```

The Index Merge intersection algorithm performs simultaneous scans on all used indexes and produces the intersection of row sequences that it receives from the merged index scans.

If all columns used in the query are covered by the used indexes, full table rows are not retrieved (EXPLAIN output contains Using index in Extra field in this case). Here is an example of such a query:

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

If the used indexes don't cover all columns used in the query, full rows are retrieved only when the range conditions for all used keys are satisfied.

If one of the merged conditions is a condition over a primary key of an InnoDB or BDB table, it is not used for row retrieval, but is used to filter out rows

retrieved using other conditions.

7.2.6.2. The Index Merge Union Access Algorithm

The applicability criteria for this algorithm are similar to those for the Index Merge method intersection algorithm. The algorithm can be employed when the table's `WHERE` clause was converted to several range conditions on different keys combined with `OR`, and each condition is one of the following:

- In this form, where the index has exactly N parts (that is, all index parts are covered):

key_part1=const1 AND key_part2=const2 ... AND key_partN=constN

- Any range condition over a primary key of an InnoDB or BDB table.
- A condition for which the Index Merge method intersection algorithm is applicable.

Examples:

```
SELECT * FROM t1 WHERE key1=1 OR key2=2 OR key3=3;
```

```
SELECT * FROM innodb_table WHERE (key1=1 AND key2=2) OR  
(key3='foo' AND key4='bar') AND key5=5;
```

7.2.6.3. The Index Merge Sort-Union Access Algorithm

This access algorithm is employed when the `WHERE` clause was converted to several range conditions combined by `OR`, but for which the Index Merge method union algorithm is not applicable.

Examples:

```
SELECT * FROM tbl_name WHERE key_col1 < 10 OR key_col2 < 20;
```

```
SELECT * FROM tbl_name  
WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col=30;
```

The difference between the sort-union algorithm and the union algorithm is that the sort-union algorithm must first fetch row IDs for all rows and sort them

before returning any rows.

7.2.7. IS NULL Optimization

MySQL can perform the same optimization on *col_name* IS NULL that it can use for *col_name* = *constant_value*. For example, MySQL can use indexes and ranges to search for NULL with IS NULL.

Examples:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;
```

```
SELECT * FROM tbl_name WHERE key_col <=> NULL;
```

```
SELECT * FROM tbl_name  
  WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a WHERE clause includes a *col_name* IS NULL condition for a column that is declared as NOT NULL, that expression is optimized away. This optimization does not occur in cases when the column might produce NULL anyway; for example, if it comes from a table on the right side of a LEFT JOIN.

MySQL can also optimize the combination *col_name* = *expr* AND *col_name* IS NULL, a form that is common in resolved subqueries. EXPLAIN shows *ref_or_null* when this optimization is used.

This optimization can handle one IS NULL for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns a and b of table t2:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;
```

```
SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;
```

```
SELECT * FROM t1, t2  
  WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;
```

```
SELECT * FROM t1, t2  
  WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);
```

```
SELECT * FROM t1, t2  
  WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)  
  OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

`ref_or_null` works by first doing a read on the reference key, and then a separate search for rows with a `NULL` key value.

Note that the optimization can handle only one `IS NULL` level. In the following query, MySQL uses key lookups only on the expression `(t1.a=t2.a AND t2.a IS NULL)` and is not able to use the key part on `b`:

```
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL)
  OR (t1.b=t2.b AND t2.b IS NULL);
```

7.2.8. DISTINCT Optimization

`DISTINCT` combined with `ORDER BY` needs a temporary table in many cases.

Because `DISTINCT` may use `GROUP BY`, you should be aware of how MySQL works with columns in `ORDER BY` or `HAVING` clauses that are not part of the selected columns. See [Section 12.10.3, “GROUP BY and HAVING with Hidden Fields”](#).

In most cases, a `DISTINCT` clause can be considered as a special case of `GROUP BY`. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1 WHERE c1 > const;
SELECT c1, c2, c3 FROM t1 WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to `GROUP BY` queries can be also applied to queries with a `DISTINCT` clause. Thus, for more details on the optimization possibilities for `DISTINCT` queries, see [Section 7.2.13, “GROUP BY Optimization”](#).

When combining `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds `row_count` unique rows.

If you do not use columns from all tables named in a query, MySQL stops scanning any unused tables as soon as it finds the first match. In the following case, assuming that `t1` is used before `t2` (which you can check with `EXPLAIN`), MySQL stops reading from `t2` (for any particular row in `t1`) when it finds the first row in `t2`:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

7.2.9. LEFT JOIN and RIGHT JOIN Optimization

MySQL implements an A LEFT JOIN B join_condition as follows:

- Table B is set to depend on table A and all tables on which A depends.
- Table A is set to depend on all tables (except B) that are used in the LEFT JOIN condition.
- The LEFT JOIN condition is used to decide how to retrieve rows from table B . (In other words, any condition in the WHERE clause is not used.)
- All standard join optimizations are performed, with the exception that a table is always read after all tables on which it depends. If there is a circular dependence, MySQL issues an error.
- All standard WHERE optimizations are performed.
- If there is a row in A that matches the WHERE clause, but there is no row in B that matches the ON condition, an extra B row is generated with all columns set to NULL.
- If you use LEFT JOIN to find rows that do not exist in some table and you have the following test: col_name IS NULL in the WHERE part, where col_name is a column that is declared as NOT NULL, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the LEFT JOIN condition.

The implementation of RIGHT JOIN is analogous to that of LEFT JOIN with the roles of the tables reversed.

The join optimizer calculates the order in which tables should be joined. The table read order forced by LEFT JOIN or STRAIGHT_JOIN helps the join optimizer do its work much more quickly, because there are fewer table permutations to check. Note that this means that if you do a query of the following type, MySQL does a full scan on b because the LEFT JOIN forces it to be read before d :

```
SELECT *
```

```
FROM a JOIN b LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a
WHERE b.key=d.key;
```

The fix in this case is reverse the order in which a and b are listed in the FROM clause:

```
SELECT *
FROM b JOIN a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a
WHERE b.key=d.key;
```

For a LEFT JOIN, if the WHERE condition is always false for the generated NULL row, the LEFT JOIN is changed to a normal join. For example, the WHERE clause would be false in the following query if t2.column1 were NULL:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it is safe to convert the query to a normal join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

This can be made faster because MySQL can use table t2 before table t1 if doing so would result in a better query plan. To force a specific table order, use STRAIGHT_JOIN.

7.2.10. Nested Join Optimization

As of MySQL 5.0.1, the syntax for expressing joins allows nested joins. The following discussion refers to the join syntax described in [Section 13.2.7.1, “JOIN Syntax”](#).

The syntax of *table_factor* is extended in comparison with the SQL Standard. The latter accepts only *table_reference*, not a list of them inside a pair of parentheses. This is a conservative extension if we consider each comma in a list of *table_reference* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, `CROSS JOIN` is a syntactic equivalent to `INNER JOIN` (they can replace each other). In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause; `CROSS JOIN` is used otherwise.

In versions of MySQL prior to 5.0.1, parentheses in *table_references* were just omitted and all join operations were grouped to the left. In general, parentheses can be ignored in join expressions containing only inner join operations.

After removing parentheses and grouping operations to the left, the join expression:

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
  ON t1.a=t2.a
```

transforms into the expression:

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
  ON t2.b=t3.b OR t2.b IS NULL
```

Yet, the two expressions are not equivalent. To see this, suppose that the tables `t1`, `t2`, and `t3` have the following state:

- Table `t1` contains rows (1), (2)
- Table `t2` contains row (1, 101)
- Table `t3` contains row (101)

In this case, the first expression returns a result set including the rows (1, 1, 101, 101), (2, NULL, NULL, NULL), whereas the second expression returns the rows (1, 1, 101, 101), (2, NULL, NULL, 101):

```
mysql> SELECT *
-> FROM t1
-> LEFT JOIN
-> (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
-> ON t1.a=t2.a;
```

a	a	b	b
1	1	101	101
2	NULL	NULL	NULL

```
mysql> SELECT *
-> FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
-> LEFT JOIN t3
-> ON t2.b=t3.b OR t2.b IS NULL;
+-----+-----+-----+-----+
| a      | a      | b      | b      |
+-----+-----+-----+-----+
| 1      | 1      | 101    | 101    |
| 2      | NULL   | NULL   | 101    |
+-----+-----+-----+-----+
```

In the following example, an outer join operation is used together with an inner join operation:

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

That expression cannot be transformed into the following expression:

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3.
```

For the given table states, the two expressions return different sets of rows:

```
mysql> SELECT *
-> FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
+-----+-----+-----+-----+
| a      | a      | b      | b      |
+-----+-----+-----+-----+
| 1      | 1      | 101    | 101    |
| 2      | NULL   | NULL   | NULL   |
+-----+-----+-----+-----+
```

```
mysql> SELECT *
-> FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
+-----+-----+-----+-----+
| a      | a      | b      | b      |
+-----+-----+-----+-----+
| 1      | 1      | 101    | 101    |
| 2      | NULL   | NULL   | 101    |
+-----+-----+-----+-----+
```

Therefore, if we omit parentheses in a join expression with outer join operators, we might change the result set for the original expression.

More exactly, we cannot ignore parentheses in the right operand of the left outer join operation and in the left operand of a right join operation. In other words, we cannot ignore parentheses for the inner table expressions of outer join

operations. Parentheses for the other operand (operand for the outer table) can be ignored.

The following expression:

```
(t1, t2) LEFT JOIN t3 ON P(t2.b, t3.b)
```

is equivalent to this expression:

```
t1, t2 LEFT JOIN t3 ON P(t2.b, t3.b)
```

for any tables *t1*, *t2*, *t3* and any condition *P* over attributes *t2.b* and *t3.b*.

Whenever the order of execution of the join operations in a join expression (*join_table*) is not from left to right, we talk about nested joins. Consider the following queries:

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
WHERE t1.a > 1
```

```
SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

Those queries are considered to contain these nested joins:

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

The nested join is formed in the first query with a left join operation, whereas in the second query it is formed with an inner join operation.

In the first query, the parentheses can be omitted: The grammatical structure of the join expression will dictate the same order of execution for join operations. For the second query, the parentheses cannot be omitted, although the join expression here can be interpreted unambiguously without them. (In our extended syntax the parentheses in (*t2*, *t3*) of the second query are required, although theoretically the query could be parsed without them: We still would have unambiguous syntactical structure for the query because `LEFT JOIN` and `ON` would play the role of the left and right delimiters for the expression (*t2*, *t3*.)

The preceding examples demonstrate these points:

- For join expressions involving only inner joins (and not outer joins), parentheses can be removed. You can remove parentheses and evaluate left to right (or, in fact, you can evaluate the tables in any order).
- The same is not true, in general, for outer joins or for outer joins mixed with inner joins. Removal of parentheses may change the result.

Queries with nested outer joins are executed in the same pipeline manner as queries with inner joins. More exactly, a variation of the nested-loop join algorithm is exploited. Recall by what algorithmic schema the nested-loop join executes a query. Suppose that we have a join query over 3 tables T_1, T_2, T_3 of the form:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
          INNER JOIN T3 ON P2(T2,T3)
WHERE P(T1, T2, T3).
```

Here, $P_1(T_1, T_2)$ and $P_2(T_2, T_3)$ are some join conditions (on expressions), whereas $P(t_1, t_2, t_3)$ is a condition over columns of tables T_1, T_2, T_3 .

The nested-loop join algorithm would execute this query in the following manner:

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1, t2, t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

The notation $t_1 || t_2 || t_3$ means “a row constructed by concatenating the columns of rows $t_1, t_2,$ and t_3 .” In some of the following examples, $NULL$ where a row name appears means that $NULL$ is used for each column of that row. For example, $t_1 || t_2 || NULL$ means “a row constructed by concatenating the columns of rows t_1 and t_2 , and $NULL$ for each column of t_3 .”

Now let's consider a query with nested outer joins:

```
SELECT * FROM T1 LEFT JOIN
          (T2 LEFT JOIN T3 ON P2(T2, T3))
```

```

        ON P1(T1, T2)
WHERE P(T1, T2, T3).

```

For this query, we modify the nested-loop pattern to get:

```

FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF P(t1,t2,NULL) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}

```

In general, for any nested loop for the first inner table in an outer join operation, a flag is introduced that is turned off before the loop and is checked after the loop. The flag is turned on when for the current row from the outer table a match from the table representing the inner operand is found. If at the end of the loop cycle the flag is still off, no match has been found for the current row of the outer table. In this case, the row is complemented by NULL values for the columns of the inner tables. The result row is passed to the final check for the output or into the next nested loop, but only if the row satisfies the join condition of all embedded outer joins.

In our example, the outer join table expressed by the following expression is embedded:

```
(T2 LEFT JOIN T3 ON P2(T2, T3))
```

Note that for the query with inner joins, the optimizer could choose a different order of nested loops, such as this one:

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

For the queries with outer joins, the optimizer can choose only such an order where loops for outer tables precede loops for inner tables. Thus, for our query with outer joins, only one nesting order is possible. For the following query, the optimizer will evaluate two different nestings:

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
WHERE P(T1,T2,T3)
```

The nestings are these:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

and:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
```

```

        t:=t1||t2||t3; OUTPUT t;
    }
    f1:=TRUE
}
}
IF (!f1) {
    IF P(t1, NULL, NULL) {
        t:=t1||NULL||NULL; OUTPUT t;
    }
}
}

```

In both nestings, T_1 must be processed in the outer loop because it is used in an outer join. T_2 and T_3 are used in an inner join, so that join must be processed in the inner loop. However, because the join is an inner join, T_2 and T_3 can be processed in either order.

When discussing the nested-loop algorithm for inner joins, we omitted some details whose impact on the performance of query execution may be huge. We did not mention so-called “pushed-down” conditions. Suppose that our WHERE condition $P(T_1, T_2, T_3)$ can be represented by a conjunctive formula:

$$P(T_1, T_2, T_3) = C_1(T_1) \text{ AND } C_2(T_2) \text{ AND } C_3(T_3).$$

In this case, MySQL actually uses the following nested-loop schema for the execution of the query with inner joins:

```

FOR each row t1 in T1 such that C1(t1) {
    FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2) {
        FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
            IF P(t1, t2, t3) {
                t:=t1||t2||t3; OUTPUT t;
            }
        }
    }
}
}

```

You see that each of the conjuncts $C_1(T_1)$, $C_2(T_2)$, $C_3(T_3)$ are pushed out of the most inner loop to the most outer loop where it can be evaluated. If $C_1(T_1)$ is a very restrictive condition, this condition pushdown may greatly reduce the number of rows from table T_1 passed to the inner loops. As a result, the execution time for the query may improve immensely.

For a query with outer joins, the WHERE condition is to be checked only after it

has been found that the current row from the outer table has a match in the inner tables. Thus, the optimization of pushing conditions out of the inner nested loops cannot be applied directly to queries with outer joins. Here we have to introduce conditional pushed-down predicates guarded by the flags that are turned on when a match has been encountered.

For our example with outer joins with:

$P(T1, T2, T3) = C1(T1) \text{ AND } C(T2) \text{ AND } C3(T3)$

the nested-loop schema using guarded pushed-down conditions looks like this:

```

FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
    such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
      BOOL f2:=FALSE;
      FOR each row t3 in T3
        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
          IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
            t:=t1||t2||t3; OUTPUT t;
          }
          f2=TRUE;
          f1=TRUE;
        }
      IF (!f2) {
        IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
          t:=t1||t2||NULL; OUTPUT t;
        }
        f1=TRUE;
      }
    }
  }
  IF (!f1 && P(t1,NULL,NULL)) {
    t:=t1||NULL||NULL; OUTPUT t;
  }
}

```

In general, pushed-down predicates can be extracted from join conditions such as $P1(T1, T2)$ and $P(T2, T3)$. In this case, a pushed-down predicate is guarded also by a flag that prevents checking the predicate for the NULL-complemented row generated by the corresponding outer join operation.

Note that access by key from one inner table to another in the same nested join is prohibited if it is induced by a predicate from the WHERE condition. (We could use conditional key access in this case, but this technique is not employed yet in

MySQL 5.0.)

7.2.11. Outer Join Simplification

Table expressions in the FROM clause of a query are simplified in many cases.

At the parser stage, queries with right outer joins operations are converted to equivalent queries containing only left join operations. In the general case, the conversion is performed according to the following rule:

$$(T1, \dots) \text{ RIGHT JOIN } (T2, \dots) \text{ ON } P(T1, \dots, T2, \dots) = \\ (T2, \dots) \text{ LEFT JOIN } (T1, \dots) \text{ ON } P(T1, \dots, T2, \dots)$$

All inner join expressions of the form `T1 INNER JOIN T2 ON P(T1, T2)` are replaced by the list `T1, T2, P(T1, T2)` being joined as a conjunct to the WHERE condition (or to the join condition of the embedding join, if there is any).

When the optimizer evaluates plans for join queries with outer join operation, it takes into consideration only the plans where, for each such operation, the outer tables are accessed before the inner tables. The optimizer options are limited because only such plans enables us to execute queries with outer joins operations by the nested loop schema.

Suppose that we have a query of the form:

```
SELECT * T1 LEFT JOIN T2 ON P1(T1, T2)
      WHERE P(T1, T2) AND R(T2)
```

with `R(T2)` narrowing greatly the number of matching rows from table `T2`. If we executed the query as it is, the optimizer would have no other choice besides to access table `T1` before table `T2` that may lead to a very inefficient execution plan.

Fortunately, MySQL converts such a query into a query without an outer join operation if the WHERE condition is null-rejected. A condition is called null-rejected for an outer join operation if it evaluates to `FALSE` or to `UNKNOWN` for any `NULL`-complemented row built for the operation.

Thus, for this outer join:

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

Conditions such as these are null-rejected:

```
T2.B IS NOT NULL,  
T2.B > 3,  
T2.C <= T1.C,  
T2.B < 2 OR T2.C > 1
```

Conditions such as these are not null-rejected:

```
T2.B IS NULL,  
T1.B < 3 OR T2.B IS NOT NULL,  
T1.B < 3 OR T2.B > 3
```

The general rules for checking whether a condition is null-rejected for an outer join operation are simple. A condition is null-rejected in the following cases:

- If it is of the form `A IS NOT NULL`, where `A` is an attribute of any of the inner tables
- If it is a predicate containing a reference to an inner table that evaluates to `UNKNOWN` when one of its arguments is `NULL`
- If it is a conjunction containing a null-rejected condition as a conjunct
- If it is a disjunction of null-rejected conditions

A condition can be null-rejected for one outer join operation in a query and not null-rejected for another. In the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A  
                LEFT JOIN T3 ON T3.B=T1.B  
WHERE T3.C > 0
```

the `WHERE` condition is null-rejected for the second outer join operation but is not null-rejected for the first one.

If the `WHERE` condition is null-rejected for an outer join operation in a query, the outer join operation is replaced by an inner join operation.

For example, the preceding query is replaced with the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A  
                INNER JOIN T3 ON T3.B=T1.B
```

```
WHERE T3.C > 0
```

For the original query, the optimizer would evaluate plans compatible with only one access order $T1, T2, T3$. For the replacing query, it additionally considers the access sequence $T3, T1, T2$.

A conversion of one outer join operation may trigger a conversion of another. Thus, the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
           LEFT JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

will be first converted to the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
           INNER JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

which is equivalent to the query:

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

Now the remaining outer join operation can be replaced by an inner join, too, because the condition $T3.B=T2.B$ is null-rejected and we get a query without outer joins at all:

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

Sometimes we succeed in replacing an embedded outer join operation, but cannot convert the embedding outer join. The following query:

```
SELECT * FROM T1 LEFT JOIN
           (T2 LEFT JOIN T3 ON T3.B=T2.B)
           ON T2.A=T1.A
WHERE T3.C > 0
```

is converted to:

```
SELECT * FROM T1 LEFT JOIN
           (T2 INNER JOIN T3 ON T3.B=T2.B)
           ON T2.A=T1.A
```

```
WHERE T3.C > 0,
```

That can be rewritten only to the form still containing the embedding outer join operation:

```
SELECT * FROM T1 LEFT JOIN
      (T2, T3)
      ON (T2.A=T1.A AND T3.B=T2.B)
WHERE T3.C > 0.
```

When trying to convert an embedded outer join operation in a query, we must take into account the join condition for the embedding outer join together with the WHERE condition. In the query:

```
SELECT * FROM T1 LEFT JOIN
      (T2 LEFT JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A AND T3.C=T1.C
WHERE T3.D > 0 OR T1.D > 0
```

the WHERE condition is not null-rejected for the embedded outer join, but the join condition of the embedding outer join `T2.A=T1.A AND T3.C=T1.C` is null-rejected. So the query can be converted to:

```
SELECT * FROM T1 LEFT JOIN
      (T2, T3)
      ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
WHERE T3.D > 0 OR T1.D > 0
```

The algorithm that converts outer join operations into inner joins was implemented in full measure, as it has been described here, in MySQL 5.0.1. MySQL 4.1 performs only some simple conversions.

7.2.12. ORDER BY Optimization

In some cases, MySQL can use an index to satisfy an ORDER BY clause without doing any extra sorting.

The index can also be used even if the ORDER BY does not match the index exactly, as long as all of the unused portions of the index and all the extra ORDER BY columns are constants in the WHERE clause. The following queries use the index to resolve the ORDER BY part:

```
SELECT * FROM t1
```

```
ORDER BY key_part1,key_part2,... ;
```

```
SELECT * FROM t1  
WHERE key_part1=constant  
ORDER BY key_part2;
```

```
SELECT * FROM t1  
ORDER BY key_part1 DESC, key_part2 DESC;
```

```
SELECT * FROM t1  
WHERE key_part1=1  
ORDER BY key_part1 DESC, key_part2 DESC;
```

In some cases, MySQL *cannot* use indexes to resolve the ORDER BY, although it still uses indexes to find the rows that match the WHERE clause. These cases include the following:

- You use ORDER BY on different keys:

```
SELECT * FROM t1 ORDER BY key1, key2;
```

- You use ORDER BY on non-consecutive parts of a key:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
```

- You mix ASC and DESC:

```
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
```

- The key used to fetch the rows is not the same as the one used in the ORDER BY:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
```

- You are joining many tables, and the columns in the ORDER BY are not all from the first non-constant table that is used to retrieve rows. (This is the first table in the EXPLAIN output that does not have a const join type.)

- You have different ORDER BY and GROUP BY expressions.

- The type of table index used does not store rows in order. For example, this is true for a HASH index in a MEMORY table.

With EXPLAIN SELECT ... ORDER BY, you can check whether MySQL can use

indexes to resolve the query. It cannot if you see `Using filesort` in the Extra column. See [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#).

A `filesort` optimization is used that records not only the sort key value and row position, but the columns required for the query as well. This avoids reading the rows twice. The `filesort` algorithm works like this:

1. Read the rows that match the `WHERE` clause.
2. For each row, record a tuple of values consisting of the sort key value and row position, and also the columns required for the query.
3. Sort the tuples by sort key value
4. Retrieve the rows in sorted order, but read the required columns directly from the sorted tuples rather than by accessing the table a second time.

This algorithm represents a significant improvement over that used in some older versions of MySQL.

To avoid a slowdown, this optimization is used only if the total size of the extra columns in the sort tuple does not exceed the value of the `max_length_for_sort_data` system variable. (A symptom of setting the value of this variable too high is that you should see high disk activity and low CPU activity.)

If you want to increase `ORDER BY` speed, check whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, you can try the following strategies:

- Increase the size of the `sort_buffer_size` variable.
- Increase the size of the `read_rnd_buffer_size` variable.
- Change `tmpdir` to point to a dedicated filesystem with large amounts of empty space. This option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (‘:’) on Unix and semicolon characters (‘;’) on Windows, NetWare, and OS/2. You can use this feature to spread the load across several directories. *Note:* The paths should be for directories in filesystems that are located on different *physical*

disks, not different partitions on the same disk.

By default, MySQL sorts all `GROUP BY col1, col2, ...` queries as if you specified `ORDER BY col1, col2, ...` in the query as well. If you include an `ORDER BY` clause explicitly that contains the same column list, MySQL optimizes it away without any speed penalty, although the sorting still occurs. If a query includes `GROUP BY` but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying `ORDER BY NULL`. For example:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

7.2.13. GROUP BY Optimization

The most general way to satisfy a `GROUP BY` clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and to avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for `GROUP BY` are that all `GROUP BY` columns reference attributes from the same index, and that the index stores its keys in order (for example, this is a `BTREE` index and not a `HASH` index). Whether use of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

There are two ways to execute a `GROUP BY` query via index access, as detailed in the following sections. In the first method, the grouping operation is applied together with all range predicates (if any). The second method first performs a range scan, and then groups the resulting tuples.

7.2.13.1. Loose index scan

The most efficient way to process `GROUP BY` is when the index is used to directly retrieve the group fields. With this access method, MySQL uses the property of some index types that the keys are ordered (for example, `BTREE`). This property enables use of lookup groups in an index without having to consider all keys in the index that satisfy all `WHERE` conditions. This access method considers only a

fraction of the keys in an index, so it is called a *loose index scan*. When there is no WHERE clause, a loose index scan reads as many keys as the number of groups, which may be a much smaller number than that of all keys. If the WHERE clause contains range predicates (see the discussion of the range join type in [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#)), a loose index scan looks up the first key of each group that satisfies the range conditions, and again reads the least possible number of keys. This is possible under the following conditions:

- The query is over a single table.
- The GROUP BY includes the first consecutive parts of the index. (If, instead of GROUP BY, the query has a DISTINCT clause, all distinct attributes refer to the beginning of the index.)
- The only aggregate functions used (if any) are MIN() and MAX(), and all of them refer to the same column.
- Any other parts of the index than those from the GROUP BY referenced in the query must be constants (that is, they must be referenced in equalities with constants), except for the argument of MIN() or MAX() functions.

The EXPLAIN output for such queries shows `Using index for group-by` in the Extra column.

The following queries fall into this category, assuming that there is an index `idx(c1, c2, c3)` on table `t1(c1, c2, c3, c4)`:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

The following queries cannot be executed with this quick select method, for the reasons given:

- There are aggregate functions other than MIN() or MAX(), for example:

```
SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
```

- The fields in the `GROUP BY` clause do not refer to the beginning of the index, as shown here:

```
SELECT c1,c2 FROM t1 GROUP BY c2, c3;
```

- The query refers to a part of a key that comes after the `GROUP BY` part, and for which there is no equality with a constant, an example being:

```
SELECT c1,c3 FROM t1 GROUP BY c1, c2;
```

7.2.13.2. Tight index scan

A tight index scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a loose index scan are not met, it is still possible to avoid creation of temporary tables for `GROUP BY` queries. If there are range conditions in the `WHERE` clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Because this method reads all keys in each range defined by the `WHERE` clause, or scans the whole index if there are no range conditions, we term it a *tight index scan*. Notice that with a tight index scan, the grouping operation is performed only after all keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that there is a constant equality condition for all columns in a query referring to parts of the key coming before or in between parts of the `GROUP BY` key. The constants from the equality conditions fill in any “gaps” in the search keys so that it is possible to form complete prefixes of the index. These index prefixes then can be used for index lookups. If we require sorting of the `GROUP BY` result, and it is possible to form search keys that are prefixes of the index, MySQL also avoids extra sorting operations because searching with prefixes in an ordered index already retrieves all the keys in order.

The following queries do not work with the loose index scan access method described earlier, but still work with the tight index scan access method (assuming that there is an index `idx(c1, c2, c3)` on table `t1(c1, c2, c3, c4)`).

- There is a gap in the `GROUP BY`, but it is covered by the condition `c2 = 'a'`:

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- The `GROUP BY` does not begin with the first part of the key, but there is a condition that provides a constant for that part:

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

7.2.14. LIMIT Optimization

In some cases, MySQL handles a query differently when you are using `LIMIT row_count` and not using `HAVING`:

- If you are selecting only a few rows with `LIMIT`, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.
- If you use `LIMIT row_count` with `ORDER BY`, MySQL ends the sorting as soon as it has found the first *row_count* rows of the sorted result, rather than sorting the entire result. If ordering is done by using an index, this is very fast. If a filesort must be done, all rows that match the query without the `LIMIT` clause must be selected, and most or all of them must be sorted, before it can be ascertained that the first *row_count* rows have been found. In either case, after the initial rows have been found, there is no need to sort any remainder of the result set, and MySQL does not do so.
- When combining `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds *row_count* unique rows.
- In some cases, a `GROUP BY` can be resolved by reading the key in order (or doing a sort on the key) and then calculating summaries until the key value changes. In this case, `LIMIT row_count` does not calculate any unnecessary `GROUP BY` values.
- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using `SQL_CALC_FOUND_ROWS`.
- `LIMIT 0` quickly returns an empty set. This can be useful for checking the validity of a query. When using one of the MySQL APIs, it can also be employed for obtaining the types of the result columns. (This trick does not work in the MySQL Monitor (the **mysql** program), which merely displays Empty set in such cases; you should instead use `SHOW COLUMNS` or

DESCRIBE for this purpose.)

- When the server uses temporary tables to resolve the query, it uses the `LIMIT row_count` clause to calculate how much space is required.

7.2.15. How to Avoid Table Scans

The output from `EXPLAIN` shows `ALL` in the `type` column when MySQL uses a table scan to resolve a query. This usually happens under the following conditions:

- The table is so small that it is faster to perform a table scan than to bother with a key lookup. This is common for tables with fewer than 10 rows and a short row length.
- There are no usable restrictions in the `ON` or `WHERE` clause for indexed columns.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See [Section 7.2.4, “WHERE Clause Optimization”](#).
- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it probably will do many key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate and the performance impact is negligible. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Use `ANALYZE TABLE tbl_name` to update the key distributions for the scanned table. See [Section 13.5.2.1, “ANALYZE TABLE Syntax”](#).
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

See [Section 13.2.7, “SELECT Syntax”](#).

- Start **mysqld** with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See [Section 5.2.2, “Server System Variables”](#).

7.2.16. Speed of **INSERT** Statements

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting row: ($1 \times$ size of row)
- Inserting indexes: ($1 \times$ number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by $\log N$, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use **INSERT** statements with multiple **VALUES** lists to insert several rows at a time. This is considerably faster (many times faster in some cases) than using separate single-row **INSERT** statements. If you are adding data to a non-empty table, you can tune the `bulk_insert_buffer_size` variable to make data insertion even faster. See [Section 5.2.2, “Server System Variables”](#).

- If you are inserting a lot of rows from different clients, you can get higher speed by using the `INSERT DELAYED` statement. See [Section 13.2.4.2, “INSERT DELAYED Syntax”](#).
- For a MyISAM table, you can use concurrent inserts to add rows at the same time that `SELECT` statements are running if there are no deleted rows in middle of the table. See [Section 7.3.3, “Concurrent Inserts”](#).
- When loading a table from a text file, use `LOAD DATA INFILE`. This is usually 20 times faster than using `INSERT` statements. See [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).
- With some extra work, it is possible to make `LOAD DATA INFILE` run even faster for a MyISAM table when the table has many indexes. Use the following procedure:
 1. Optionally create the table with `CREATE TABLE`.
 2. Execute a `FLUSH TABLES` statement or a **`mysqladmin flush-tables`** command.
 3. Use **`myisamchk --keys-used=0 -rq /path/to/db/tbl_name`**. This removes all use of indexes for the table.
 4. Insert data into the table with `LOAD DATA INFILE`. This does not update any indexes and therefore is very fast.
 5. If you intend only to read from the table in the future, use **`myisampack`** to compress it. See [Section 14.1.3.3, “Compressed Table Characteristics”](#).
 6. Re-create the indexes with **`myisamchk -rq /path/to/db/tbl_name`**. This creates the index tree in memory before writing it to disk, which is much faster than updating the index during `LOAD DATA INFILE` because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.
 7. Execute a `FLUSH TABLES` statement or a **`mysqladmin flush-tables`** command.

Note that `LOAD DATA INFILE` performs the preceding optimization automatically if the MyISAM table into which you insert data is empty. The main difference is that you can let **myisamchk** allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the `LOAD DATA INFILE` statement.

You can also disable or enable the indexes for a MyISAM table by using the following statements rather than **myisamchk**. If you use these statements, you can skip the `FLUSH TABLE` operations:

```
ALTER TABLE tbl_name DISABLE KEYS;  
ALTER TABLE tbl_name ENABLE KEYS;
```

- To speed up `INSERT` operations that are performed with multiple statements for non-transactional tables, lock your tables:

```
LOCK TABLES a WRITE;  
INSERT INTO a VALUES (1, 23), (2, 34), (4, 33);  
INSERT INTO a VALUES (8, 26), (6, 29);  
...  
UNLOCK TABLES;
```

This benefits performance because the index buffer is flushed to disk only once, after all `INSERT` statements have completed. Normally, there would be as many index buffer flushes as there are `INSERT` statements. Explicit locking statements are not needed if you can insert all rows with a single `INSERT`.

To obtain faster insertions, for transactional tables, you should use `START TRANSACTION` and `COMMIT` instead of `LOCK TABLES`.

Locking also lowers the total time for multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. For example:

1. Connection 1 does 1000 inserts
2. Connections 2, 3, and 4 do 1 insert
3. Connection 5 does 1000 inserts

If you do not use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

INSERT, UPDATE, and DELETE operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five inserts or updates in a row. If you do very many inserts in a row, you could do a LOCK TABLES followed by an UNLOCK TABLES once in a while (each 1,000 rows or so) to allow other threads access to the table. This would still result in a nice performance gain.

INSERT is still much slower for loading data than LOAD DATA INFILE, even when using the strategies just outlined.

- To increase performance for MyISAM tables, for both LOAD DATA INFILE and INSERT, enlarge the key cache by increasing the key_buffer_size system variable. See [Section 7.5.2, “Tuning Server Parameters”](#).

7.2.17. Speed of UPDATE Statements

An update statement is optimized like a SELECT query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed do not get updated.

Another way to get fast updates is to delay updates and then do many updates in a row later. Performing multiple updates together is much quicker than doing one at a time if you lock the table.

For a MyISAM table that uses dynamic row format, updating a row to a longer total length may split the row. If you do this often, it is very important to use OPTIMIZE TABLE occasionally. See [Section 13.5.2.5, “OPTIMIZE TABLE Syntax”](#).

7.2.18. Speed of DELETE Statements

The time required to delete individual rows is exactly proportional to the number of indexes. To delete rows more quickly, you can increase the size of the key cache by increasing the key_buffer_size system variable. See [Section 7.5.2, “Tuning Server Parameters”](#).

To delete all rows from a table, `TRUNCATE TABLE tbl_name` is faster than `DELETE FROM tbl_name`. See [Section 13.2.9, “TRUNCATE Syntax”](#).

7.2.19. Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- Use persistent connections to the database to avoid connection overhead. If you cannot use persistent connections and you are initiating many new connections to the database, you may want to change the value of the `thread_cache_size` variable. See [Section 7.5.2, “Tuning Server Parameters”](#).
- Always check whether all your queries really use the indexes that you have created in the tables. In MySQL, you can do this with the `EXPLAIN` statement. See [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#).
- Try to avoid complex `SELECT` queries on MyISAM tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.
- With MyISAM tables that have no deleted rows in the middle, you can insert rows at the end at the same time that another query is reading from the table. If it is important to be able to do this, you should consider using the table in ways that avoid deleting rows. Another possibility is to run `OPTIMIZE TABLE` to defragment the table after you have deleted a lot of rows from it. See [Section 14.1, “The MyISAM Storage Engine”](#).
- To fix any compression issues that may have occurred with ARCHIVE tables, you can use `OPTIMIZE TABLE`. See [Section 14.8, “The ARCHIVE Storage Engine”](#).
- Use `ALTER TABLE ... ORDER BY expr1, expr2, ...` if you usually retrieve rows in `expr1, expr2, ...` order. By using this option after extensive changes to the table, you may be able to get higher performance.
- In some cases, it may make sense to introduce a column that is “hashed” based on information from other columns. If this column is short and

reasonably unique, it may be much faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:

```
SELECT * FROM tbl_name
  WHERE hash_col=MD5(CONCAT(col1,col2))
  AND col1='constant' AND col2='constant';
```

- For MyISAM tables that change frequently, you should try to avoid all variable-length columns (VARCHAR, BLOB, and TEXT). The table uses dynamic row format if it includes even a single variable-length column. See [Chapter 14, Storage Engines and Table Types](#).
- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a MyISAM table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See [Chapter 14, Storage Engines and Table Types](#).
- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

```
UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant
```

This is very important when you use MySQL storage engines such as MyISAM that has only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.

- If you need to collect statistics from large log tables, use summary tables instead of scanning the entire log table. Maintaining the summaries should be much faster than trying to calculate statistics “live.” Regenerating new summary tables from the logs when things change (depending on business decisions) is faster than changing the running application.
- If possible, you should classify reports as “live” or as “statistical,” where data needed for statistical reports is created only from summary tables that

are generated periodically from the live data.

- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.
- In some cases, it is convenient to pack and store data into a BLOB column. In this case, you must provide code in your application to pack and unpack information, but this may save a lot of accesses at some stage. This is practical when you have data that does not conform well to a rows-and-columns table structure.
- Normally, you should try to keep all data non-redundant (observing what is referred to in database theory as *third normal form*). However, there may be situations in which it can be advantageous to duplicate information or create summary tables to gain more speed.
- Stored routines or UDFs (user-defined functions) may be a good way to gain performance for some tasks. See [Chapter 17, *Stored Procedures and Functions*](#), and [Section 24.2, “Adding New Functions to MySQL”](#), for more information.
- You can always gain something by caching queries or answers in your application and then performing many inserts or updates together. If your database system supports table locks (as do MySQL and Oracle), this should help to ensure that the index cache is only flushed once after all updates. You can also take advantage of MySQL's query cache to achieve similar results; see [Section 5.14, “The MySQL Query Cache”](#).
- Use `INSERT DELAYED` when you do not need to know when your data is written. This reduces the overall insertion impact because many rows can be written with a single disk write.
- Use `INSERT LOW_PRIORITY` when you want to give `SELECT` statements higher priority than your inserts.
- Use `SELECT HIGH_PRIORITY` to get retrievals that jump the queue. That is, the `SELECT` is executed even if there is another client waiting to do a write.
- Use multiple-row `INSERT` statements to store many rows with one SQL

statement. Many SQL servers support this, including MySQL.

- Use `LOAD DATA INFILE` to load large amounts of data. This is faster than using `INSERT` statements.
- Use `AUTO_INCREMENT` columns to generate unique values.
- Use `OPTIMIZE TABLE` once in a while to avoid fragmentation with dynamic-format MyISAM tables. See [Section 14.1.3, “MyISAM Table Storage Formats”](#).
- Use `MEMORY (HEAP)` tables when possible to get more speed. See [Section 14.4, “The MEMORY \(HEAP\) Storage Engine”](#). `MEMORY` tables are useful for non-critical data that is accessed often, such as information about the last displayed banner for users who don't have cookies enabled in their Web browser. User sessions are another alternative available in many Web application environments for handling volatile state data.
- With Web servers, images and other binary assets should normally be stored as files. That is, store only a reference to the file rather than the file itself in the database. Most Web servers are better at caching files than database contents, so using files is generally faster.
- Columns with identical information in different tables should be declared to have identical data types so that joins based on the corresponding columns will be faster.
- Try to keep column names simple. For example, in a table named `customer`, use a column name of `name` instead of `customer_name`. To make your names portable to other SQL servers, you should keep them shorter than 18 characters.
- If you need really high speed, you should take a look at the low-level interfaces for data storage that the different SQL servers support. For example, by accessing the MySQL `MyISAM` storage engine directly, you could get a speed increase of two to five times compared to using the SQL interface. To be able to do this, the data must be on the same server as the application, and usually it should only be accessed by one process (because external file locking is really slow). One could eliminate these problems by introducing low-level `MyISAM` commands in the MySQL server (this could be one easy way to get more performance if needed). By carefully

designing the database interface, it should be quite easy to support this type of optimization.

- If you are using numerical data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you need not parse your text files to find line and column boundaries.
- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replication servers to split up the load. To avoid slowing down the master while making backups, you can make backups using a slave server. See [Chapter 6, Replication](#).
- Declaring a MyISAM table with the `DELAY_KEY_WRITE=1` table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you should ensure that the table is okay by running the server with the `--myisam-recover` option, or by running **myisamchk** before restarting the server. (However, even in this case, you should not lose anything by using `DELAY_KEY_WRITE`, because the key information can always be generated from the data rows.)

7.3. Locking Issues

7.3.1. Locking Methods

MySQL uses table-level locking for `MyISAM` and `MEMORY` tables, page-level locking for `BDB` tables, and row-level locking for `InnoDB` tables.

In many cases, you can make an educated guess about which locking type is best for an application, but generally it is difficult to say that a given lock type is better than another. Everything depends on the application and different parts of an application may require different lock types.

To decide whether you want to use a storage engine with row-level locking, you should look at what your application does and what mix of select and update statements it uses. For example, most Web applications perform many selects, relatively few deletes, updates based mainly on key values, and inserts into a few specific tables. The base MySQL `MyISAM` setup is very well tuned for this.

Table locking in MySQL is deadlock-free for storage engines that use table-level locking. Deadlock avoidance is managed by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order.

The table-locking method MySQL uses for `WRITE` locks works as follows:

- If there are no locks on the table, put a write lock on it.
- Otherwise, put the lock request in the write lock queue.

The table-locking method MySQL uses for `READ` locks works as follows:

- If there are no write locks on the table, put a read lock on it.
- Otherwise, put the lock request in the read lock queue.

When a lock is released, the lock is made available to the threads in the write lock queue and then to the threads in the read lock queue. This means that if you have many updates for a table, `SELECT` statements wait until there are no more

updates.

You can analyze the table lock contention on your system by checking the `Table_locks_waited` and `Table_locks_immediate` status variables:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----+-----+
| Variable_name      | Value      |
+-----+-----+
| Table_locks_immediate | 1151552   |
| Table_locks_waited   | 15324     |
+-----+-----+
```

If a MyISAM table contains no free blocks in the middle, rows always are inserted at the end of the data file. In this case, you can freely mix concurrent `INSERT` and `SELECT` statements for a MyISAM table without locks. That is, you can insert rows into a MyISAM table at the same time other clients are reading from it. (Holes can result from rows having been deleted from or updated in the middle of the table. If there are holes, concurrent inserts are disabled but are re-enabled automatically when all holes have been filled with new data.)

If you want to perform many `INSERT` and `SELECT` operations on a table when concurrent inserts are not possible, you can insert rows in a temporary table and update the real table with the rows from the temporary table once in a while. This can be done with the following code:

```
mysql> LOCK TABLES real_table WRITE, insert_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM insert_table;
mysql> TRUNCATE TABLE insert_table;
mysql> UNLOCK TABLES;
```

InnoDB uses row locks and BDB uses page locks. For these two storage engines, deadlocks are possible because they automatically acquire locks during the processing of SQL statements, not at the start of the transaction.

Advantages of row-level locking:

- Fewer lock conflicts when accessing different rows in many threads.
- Fewer changes for rollbacks.
- Possible to lock a single row for a long time.

Disadvantages of row-level locking:

- Requires more memory than page-level or table-level locks.
- Slower than page-level or table-level locks when used on a large part of the table because you must acquire many more locks.
- Definitely much slower than other locks if you often do `GROUP BY` operations on a large part of the data or if you must scan the entire table frequently.

Table locks are superior to page-level or row-level locks in the following cases:

- Most statements for the table are reads.
- A mix of reads and writes, where writes are updates or deletes for a single row that can be fetched with one key read:

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;  
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- `SELECT` combined with concurrent `INSERT` statements, and very few `UPDATE` or `DELETE` statements.
- Many scans or `GROUP BY` operations on the entire table without any writers.

With higher-level locks, you can more easily tune applications by supporting locks of different types, because the lock overhead is less than for row-level locks.

Options other than row-level or page-level locking:

- Versioning (such as that used in MySQL for concurrent inserts) where it is possible to have one writer at the same time as many readers. This means that the database or table supports different views for the data depending on when access begins. Other common terms for this are “time travel,” “copy on write,” or “copy on demand.”
- Copy on demand is in many cases superior to page-level or row-level locking. However, in the worst case, it can use much more memory than using normal locks.

- Instead of using row-level locks, you can employ application-level locks, such as `GET_LOCK()` and `RELEASE_LOCK()` in MySQL. These are advisory locks, so they work only in well-behaved applications. (See [Section 12.9.4, “Miscellaneous Functions”](#).)

7.3.2. Table Locking Issues

To achieve a very high lock speed, MySQL uses table locking (instead of page, row, or column locking) for all storage engines except InnoDB and BDB.

For InnoDB and BDB tables, MySQL uses only table locking if you explicitly lock the table with `LOCK TABLES`. For these storage engines, we recommend that you not use `LOCK TABLES` at all, because InnoDB uses automatic row-level locking and BDB uses page-level locking to ensure transaction isolation.

For large tables, table locking is much better than row locking for most applications, but there are some pitfalls:

- Table locking enables many threads to read from a table at the same time, but if a thread wants to write to a table, it must first get exclusive access. During the update, all other threads that want to access this particular table must wait until the update is done.
- Table updates normally are considered to be more important than table retrievals, so they are given higher priority. This should ensure that updates to a table are not “starved” even if there is heavy `SELECT` activity for the table.
- Table locking causes problems in cases such as when a thread is waiting because the disk is full and free space needs to become available before the thread can proceed. In this case, all threads that want to access the problem table are also put in a waiting state until more disk space is made available.

Table locking is also disadvantageous under the following scenario:

- A client issues a `SELECT` that takes a long time to run.
- Another client then issues an `UPDATE` on the same table. This client waits until the `SELECT` is finished.

- Another client issues another SELECT statement on the same table. Because UPDATE has higher priority than SELECT, this SELECT waits for the UPDATE to finish, *and* for the first SELECT to finish.

The following items describe some ways to avoid or reduce contention caused by table locking:

- Try to get the SELECT statements to run faster so that they lock tables for a shorter time. You might have to create some summary tables to do this.
- Start **mysqld** with `--low-priority-updates`. This gives all statements that update (modify) a table lower priority than SELECT statements. In this case, the second SELECT statement in the preceding scenario would execute before the UPDATE statement, and would not need to wait for the first SELECT to finish.
- You can specify that all updates issued in a specific connection should be done with low priority by using the `SET LOW_PRIORITY_UPDATES=1` statement. See [Section 13.5.3, “SET Syntax”](#).
- You can give a specific INSERT, UPDATE, or DELETE statement lower priority with the `LOW_PRIORITY` attribute.
- You can give a specific SELECT statement higher priority with the `HIGH_PRIORITY` attribute. See [Section 13.2.7, “SELECT Syntax”](#).
- You can start **mysqld** with a low value for the `max_write_lock_count` system variable to force MySQL to temporarily elevate the priority of all SELECT statements that are waiting for a table after a specific number of inserts to the table occur. This allows READ locks after a certain number of WRITE locks.
- If you have problems with INSERT combined with SELECT, you might want to consider switching to MyISAM tables, which support concurrent SELECT and INSERT statements. (See [Section 7.3.3, “Concurrent Inserts”](#).)
- If you mix inserts and deletes on the same table, `INSERT DELAYED` may be of great help. See [Section 13.2.4.2, “INSERT DELAYED Syntax”](#).
- If you have problems with mixed SELECT and DELETE statements, the LIMIT

option to DELETE may help. See [Section 13.2.1, “DELETE Syntax”](#).

- Using `SQL_BUFFER_RESULT` with `SELECT` statements can help to make the duration of table locks shorter. See [Section 13.2.7, “SELECT Syntax”](#).
- You could change the locking code in `mysys/thr_lock.c` to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

Here are some tips concerning table locks in MySQL:

- Concurrent users are not a problem if you do not mix updates with selects that need to examine many rows in the same table.
- You can use `LOCK TABLES` to increase speed, because many updates within a single lock is much faster than updating without locks. Splitting table contents into separate tables may also help.
- If you encounter speed problems with table locks in MySQL, you may be able to improve performance by converting some of your tables to InnoDB or BDB tables. See [Section 14.2, “The InnoDB Storage Engine”](#), and [Section 14.5, “The BDB \(BerkeleyDB\) Storage Engine”](#).

7.3.3. Concurrent Inserts

For a MyISAM table, you can use concurrent inserts to add rows at the same time that `SELECT` statements are running if there are no deleted rows in middle of the table.

Under circumstances where concurrent inserts can be used, there is seldom any need to use the `DELAYED` modifier for `INSERT` statements. See [Section 13.2.4.2, “INSERT DELAYED Syntax”](#).

If you are using the binary log, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation.

With `LOAD DATA INFILE`, if you specify `CONCURRENT` with a MyISAM table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in

the middle), other threads can retrieve data from the table while LOAD DATA is executing. Using this option affects the performance of LOAD DATA a bit, even if no other thread is using the table at the same time.

7.4. Optimizing Database Structure

7.4.1. Design Choices

MySQL keeps row data and index data in separate files. Many (almost all) other database systems mix row and index data in the same file. We believe that the MySQL choice is better for a very wide range of modern systems.

Another way to store the row data is to keep the information for each column in a separate area (examples are SDBM and Focus). This causes a performance hit for every query that accesses more than one column. Because this degenerates so quickly when more than one column is accessed, we believe that this model is not good for general-purpose databases.

The more common case is that the index and data are stored together (as in Oracle/Sybase, et al). In this case, you find the row information at the leaf page of the index. The good thing with this layout is that it, in many cases, depending on how well the index is cached, saves a disk read. The bad things with this layout are:

- Table scanning is much slower because you have to read through the indexes to get at the data.
- You cannot use only the index table to retrieve data for a query.
- You use more space because you must duplicate indexes from the nodes (you cannot store the row in the nodes).
- Deletes degenerate the table over time (because indexes in nodes are usually not updated on delete).
- It is more difficult to cache only the index data.

7.4.2. Make Your Data as Small as Possible

One of the most basic optimizations is to design your tables to take as little space on the disk as possible. This can result in huge improvements because disk reads are faster, and smaller tables normally require less main memory while their

contents are being actively processed during query execution. Indexing also is a lesser resource burden if done on smaller columns.

MySQL supports many different storage engines (table types) and row formats. For each table, you can decide which storage and indexing method to use. Choosing the proper table format for your application may give you a big performance gain. See [Chapter 14, Storage Engines and Table Types](#).

You can get better performance for a table and minimize storage space by using the techniques listed here:

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory. For example, use the smaller integer types if possible to get smaller tables. `MEDIUMINT` is often a better choice than `INT` because a `MEDIUMINT` column uses 25% less space.
- Declare columns to be `NOT NULL` if possible. It makes everything faster and you save one bit per column. If you really need `NULL` in your application, you should definitely use it. Just avoid having it on all columns by default.
- For `MyISAM` tables, if you do not have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size row format is used. This is faster but unfortunately may waste some space. See [Section 14.1.3, “MyISAM Table Storage Formats”](#). You can hint that you want to have fixed length rows even if you have `VARCHAR` columns with the `CREATE TABLE` option `ROW_FORMAT=FIXED`.
- Starting with MySQL 5.0.3, `InnoDB` tables use a more compact storage format. In earlier versions of MySQL, `InnoDB` rows contain some redundant information, such as the number of columns and the length of each column, even for fixed-size columns. By default, tables are created in the compact format (`ROW_FORMAT=COMPACT`). If you wish to downgrade to older versions of MySQL, you can request the old format with `ROW_FORMAT=REDUNDANT`.

The compact `InnoDB` format also changes how `CHAR` columns containing UTF-8 data are stored. With `ROW_FORMAT=REDUNDANT`, a UTF-8 `CHAR(N)` occupies $3 \times N$ bytes, given that the maximum length of a UTF-8 encoded character is three bytes. Many languages can be written primarily using single-byte UTF-8 characters, so a fixed storage length often wastes space. With `ROW_FORMAT=COMPACT` format, `InnoDB` allocates a variable amount of

storage in the range from N to $3 \times N$ bytes for these columns by stripping trailing spaces if necessary. The minimum storage length is kept as N bytes to facilitate in-place updates in typical cases.

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient.
- Create only the indexes that you really need. Indexes are good for retrieval but bad when you need to store data quickly. If you access a table mostly by searching on a combination of columns, create an index on them. The first part of the index should be the column most used. If you *always* use many columns when selecting from the table, you should use the column with more duplicates first to obtain better compression of the index.
- If it is very likely that a string column has a unique prefix on the first number of characters, it's better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see [Section 13.1.4, “CREATE INDEX Syntax”](#)). Shorter indexes are faster, not only because they require less disk space, but because they give also you more hits in the index cache, and thus fewer disk seeks. See [Section 7.5.2, “Tuning Server Parameters”](#).
- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic-format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.

7.4.3. Column Indexes

All MySQL data types can be indexed. Use of indexes on the relevant columns is the best way to improve the performance of SELECT operations.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See [Chapter 14, Storage Engines and Table Types](#). All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

With `col_name(N)` syntax in an index specification, you can create an index that uses only the first N characters of a string column. Indexing only a prefix of

column values in this way can make the index file much smaller. When you index a BLOB or TEXT column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for InnoDB tables). Note that prefix limits are measured in bytes, whereas the prefix length in CREATE TABLE statements is interpreted as number of characters. *Be sure to take this into account when specifying a prefix length for a column that uses a multi-byte character set.*

You can also create FULLTEXT indexes. These are used for full-text searches. Only the MyISAM storage engine supports FULLTEXT indexes and only for CHAR, VARCHAR, and TEXT columns. Indexing always takes place over the entire column and partial (column prefix) indexing is not supported. For details, see [Section 12.7, “Full-Text Search Functions”](#).

You can also create indexes on spatial data types. Currently, only MyISAM supports R-tree indexes on spatial types. As of MySQL 5.0.16, other storage engines use B-trees for indexing spatial types (except for ARCHIVE and NDBCLUSTER, which do not support spatial type indexing).

The MEMORY storage engine uses HASH indexes by default, but also supports BTREE indexes.

7.4.4. Multiple-Column Indexes

MySQL can create composite indexes (that is, indexes on multiple columns). An index may consist of up to 15 columns. For certain data types, you can index a prefix of the column (see [Section 7.4.3, “Column Indexes”](#)).

A multiple-column index can be considered a sorted array containing values that are created by concatenating the values of the indexed columns.

MySQL uses multiple-column indexes in such a way that queries are fast when you specify a known quantity for the first column of the index in a WHERE clause, even if you do not specify values for the other columns.

Suppose that a table has the following specification:

```

CREATE TABLE test (
    id          INT NOT NULL,
    last_name   CHAR(30) NOT NULL,
    first_name  CHAR(30) NOT NULL,
    PRIMARY KEY (id),
    INDEX name (last_name,first_name)
);

```

The name index is an index over the `last_name` and `first_name` columns. The index can be used for queries that specify values in a known range for `last_name`, or for both `last_name` and `first_name`. Therefore, the name index is used in the following queries:

```
SELECT * FROM test WHERE last_name='Widenius';
```

```
SELECT * FROM test
WHERE last_name='Widenius' AND first_name='Michael';
```

```
SELECT * FROM test
WHERE last_name='Widenius'
AND (first_name='Michael' OR first_name='Monty');
```

```
SELECT * FROM test
WHERE last_name='Widenius'
AND first_name >='M' AND first_name < 'N';
```

However, the name index is *not* used in the following queries:

```
SELECT * FROM test WHERE first_name='Michael';
```

```
SELECT * FROM test
WHERE last_name='Widenius' OR first_name='Michael';
```

The manner in which MySQL uses indexes to improve query performance is discussed further in [Section 7.4.5, “How MySQL Uses Indexes”](#).

7.4.5. How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. If a table has 1,000 rows, this is at least 100 times faster than reading

sequentially. If you need to access most of the rows, it is faster to read sequentially, because this minimizes disk seeks.

Most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) are stored in B-trees. Exceptions are that indexes on spatial data types use R-trees, and that MEMORY tables also support hash indexes.

Strings are automatically prefix- and end-space compressed. See [Section 13.1.4, “CREATE INDEX Syntax”](#).

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in MEMORY tables) are described at the end of this section.

MySQL uses indexes for these operations:

- To find the rows matching a WHERE clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows.
- To retrieve rows from other tables when performing joins.
- To find the MIN() or MAX() value for a specific indexed column *key_col*. This is optimized by a preprocessor that checks whether you are using WHERE *key_part_N* = *constant* on all key parts that occur before *key_col* in the index. In this case, MySQL does a single key lookup for each MIN() or MAX() expression and replaces it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

```
SELECT MIN(key_part2), MAX(key_part2)  
FROM tbl_name WHERE key_part1=10;
```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable key (for example, ORDER BY *key_part1*, *key_part2*). If all key parts are followed by DESC, the key is read in reverse order. See [Section 7.2.12, “ORDER BY Optimization”](#).
- In some cases, a query can be optimized to retrieve values without consulting the data rows. If a query uses only columns from a table that are

numeric and that form a leftmost prefix for some key, the selected values may be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name
WHERE key_part1=1
```

Suppose that you issue the following SELECT statement:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on col1 and col2, the appropriate rows can be fetched directly. If separate single-column indexes exist on col1 and col2, the optimizer tries to find the most restrictive index by deciding which index finds fewer rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to find rows. For example, if you have a three-column index on (col1, col2, col3), you have indexed search capabilities on (col1), (col1, col2), and (col1, col2, col3).

MySQL cannot use a partial index if the columns do not form a leftmost prefix of the index. Suppose that you have the SELECT statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on (col1, col2, col3), only the first two queries use the index. The third and fourth queries do involve indexed columns, but (col2) and (col2, col3) are not leftmost prefixes of (col1, col2, col3).

A B-tree index can be used for column comparisons in expressions that use the =, >, >=, <, <=, or BETWEEN operators. The index also can be used for LIKE comparisons if the argument to LIKE is a constant string that does not start with a wildcard character. For example, the following SELECT statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

In the first statement, only rows with 'Patrick' <= key_col < 'Patrickl' are

considered. In the second statement, only rows with 'Pat' <= key_col < 'Pau' are considered.

The following SELECT statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

In the first statement, the LIKE value begins with a wildcard character. In the second statement, the LIKE value is not a constant.

If you use ... LIKE '%string%' and *string* is longer than three characters, MySQL uses the *Turbo Boyer-Moore algorithm* to initialize the pattern for the string and then uses this pattern to perform the search more quickly.

A search using col_name IS NULL employs indexes if *col_name* is indexed.

Any index that does not span all AND levels in the WHERE clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every AND group.

The following WHERE clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3
/* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2
/* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5
/* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

These WHERE clauses do *not* use indexes:

```
/* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

/* Index is not used in both parts of the WHERE clause */
... WHERE index=1 OR A=10

/* No index spans all rows */
... WHERE index_part1=1 OR index_part2=10
```

Sometimes MySQL does not use an index, even if one is available. One circumstance under which this occurs is when the optimizer estimates that using

the index would require MySQL to access a very large percentage of the rows in the table. (In this case, a table scan is likely to be much faster because it requires fewer seeks.) However, if such a query uses `LIMIT` to retrieve only some of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash indexes have somewhat different characteristics from those just discussed:

- They are used only for equality comparisons that use the `=` or `<=>` operators (but are *very* fast). They are not used for comparison operators such as `<` that find a range of values.
- The optimizer cannot use a hash index to speed up `ORDER BY` operations. (This type of index cannot be used to search for the next entry in order.)
- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a `MyISAM` table to a hash-indexed `MEMORY` table.
- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

7.4.6. The `MyISAM` Key Cache

To minimize disk I/O, the `MyISAM` storage engine exploits a strategy that is used by many database management systems. It employs a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (or *key buffer*) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.
- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system filesystem cache.

This section first describes the basic operation of the `MyISAM` key cache. Then it discusses features that improve key cache performance and that enable you to better control cache operation:

- Access to the key cache no longer is serialized among threads. Multiple threads can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

To control the size of the key cache, use the `key_buffer_size` system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the `key_buffer_size` value is too small to allocate the minimal number of block buffers (8).

When the key cache is not operational, index files are accessed using only the native filesystem buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the MyISAM index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are non-leaf nodes.)

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one of these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks) and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered “dirty.” In this case, prior to being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an *LRU (Least Recently Used)* strategy: When choosing a block for replacement, it selects the least recently used index block.

To make this choice easier, the key cache module maintains a special queue (*LRU chain*) of all used blocks. When a block is accessed, it is placed at the end of the queue. When blocks need to be replaced, blocks at the beginning of the queue are the least recently used and become the first candidates for eviction.

7.4.6.1. Shared Key Cache Access

Threads can access key cache buffers simultaneously, subject to the following conditions:

- A buffer that is not being updated can be accessed by multiple threads.
- A buffer that is being updated causes threads that need to use it to wait until the update is complete.
- Multiple threads can initiate requests that result in cache block replacements, as long as they do not interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache enables the server to improve throughput significantly.

7.4.6.2. Multiple Key Caches

Shared access to the key cache improves performance but does not eliminate contention among threads entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL also provides multiple key caches. This feature enables you to assign different table indexes to different key caches.

Where there are multiple key caches, the server must know which cache to use when processing queries for a given MyISAM table. By default, all MyISAM table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the `CACHE INDEX` statement (see [Section 13.5.5.1, “CACHE INDEX Syntax”](#)). For example, the following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
```

Table	Op	Msg_type	Msg_text
test.t1	assign_to_keycache	status	OK
test.t2	assign_to_keycache	status	OK
test.t3	assign_to_keycache	status	OK

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a `SET GLOBAL` parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

Note that you cannot destroy the default key cache. Any attempt to do this will be ignored:

```
mysql> SET GLOBAL key_buffer_size = 0;
```

```
mysql> SHOW VARIABLES LIKE 'key_buffer_size';
```

Variable_name	Value
key_buffer_size	8384512

Key cache variables are structured system variables that have a name and components. For `keycache1.key_buffer_size`, `keycache1` is the cache variable name and `key_buffer_size` is the cache component. See [Section 5.2.3.1, “Structured System Variables”](#), for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, we recommend a strategy that uses three key caches:

- A “hot” key cache that takes up 20% of the space allocated for all key caches. Use this for tables that are heavily used for searches but that are not

updated.

- A “cold” key cache that takes up 20% of the space allocated for all key caches. Use this cache for medium-sized, intensively modified tables, such as temporary tables.
- A “warm” key cache that takes up 60% of the key cache space. Employ this as the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Statements that access tables assigned to one cache do not compete with statements that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.
- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to non-leaf nodes of the index B-tree remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read in from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

CACHE INDEX sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an `init-file` option that names a file containing CACHE INDEX statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysqld_init.sql
```

The statements in `mysqld_init.sql` are executed each time the server starts. The file should contain one SQL statement per line. The following example assigns several tables each to `hot_cache` and `cold_cache`:

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

7.4.6.3. Midpoint Insertion Strategy

By default, the key cache management system uses the LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the *midpoint insertion strategy*.

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sub-chain and a warm sub-chain. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not “too short,” always containing at least `key_cache_division_limit` percent of the key cache blocks. `key_cache_division_limit` is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sub-chain. After a certain number of hits (accesses of the block), it is promoted to the hot sub-chain. At present, the number of hits required to promote a block (3) is the same for all index blocks.

A block promoted into the hot sub-chain is placed at the end of the chain. The block then circulates within this sub-chain. If the block stays at the beginning of the sub-chain for a long enough time, it is demoted to the warm chain. This time is determined by the value of the `key_cache_age_threshold` component of the key cache.

The threshold value prescribes that, for a key cache containing N blocks, the block at the beginning of the hot sub-chain not accessed within the last $N \times \text{key_cache_age_threshold} / 100$ hits is to be moved to the beginning of the warm sub-chain. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sub-chain.

The midpoint insertion strategy allows you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the

key_cache_division_limit value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the key_cache_division_limit set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sub-chain during an index scan operation as well.

7.4.6.4. Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its non-leaf nodes, it makes sense to preload the key cache with index blocks before starting to use it. Preloading allows you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks will stay in the cache, because there are enough buffers for all of them, they are fetched from disk in random order, and not sequentially.

To preload an index into a cache, use the `LOAD INDEX INTO CACHE` statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+-----+-----+-----+-----+
| Table   | Op           | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+-----+-----+-----+-----+
```

The `IGNORE LEAVES` modifier causes only blocks for the non-leaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from `t1`, but only blocks for the non-leaf nodes from `t2`.

If an index has been assigned to a key cache using a `CACHE INDEX` statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

7.4.6.5. Key Cache Block Size

It is possible to specify the size of the block buffers for an individual key cache using the `key_cache_block_size` variable. This permits tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

Currently, you cannot control the size of the index blocks in a table. This size is set by the server when the `.MYI` index file is created, depending on the size of the keys in the indexes present in the table definition. In most cases, it is set equal to the I/O buffer size.

7.4.6.6. Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the `key_buffer_size` or `key_cache_block_size` key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you change other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native filesystem caching. Filesystem caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. After the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of filesystem caching for the indexes ceases.

7.4.7. MyISAM Index Statistics Collection

Storage engines collect statistics about tables for use by the optimizer. Table statistics are based on value groups, where a value group is a set of rows with the same key prefix value. For optimizer purposes, an important statistic is the average value group size.

MySQL uses the average value group size in the following ways:

- To estimate how many rows must be read for each `ref` access
- To estimate how many row a partial join will produce; that is, the number of rows that an operation of this form will produce:

```
(...) JOIN tbl_name ON tbl_name.key = expr
```

As the average value group size for an index increases, the index is less useful for those two purposes because the average number of rows per lookup increases: For the index to be good for optimization purposes, it is best that each index value target a small number of rows in the table. When a given index value yields a large number of rows, the index is less useful and MySQL is less likely to use it.

The average value group size is related to table cardinality, which is the number of value groups. The `SHOW INDEX` statement displays a cardinality value based on N/S , where N is the number of rows in the table and S is the average value group size. That ratio yields an approximate number of value groups in the table.

For a join based on the `<=>` comparison operator, `NULL` is not treated differently from any other value: `NULL <=> NULL`, just as `N <=> N` for any other N .

However, for a join based on the `=` operator, `NULL` is different from non-`NULL` values: `expr1 = expr2` is not true when `expr1` or `expr2` (or both) are `NULL`. This affects `ref` accesses for comparisons of the form `tbl_name.key = expr`: MySQL will not access the table if the current value of `expr` is `NULL`, because the comparison cannot be true.

For `=` comparisons, it does not matter how many `NULL` values are in the table. For optimization purposes, the relevant value is the average size of the non-`NULL` value groups. However, MySQL does not currently allow that average size to be

collected or used.

For MyISAM tables, you have some control over collection of table statistics by means of the `myisam_stats_method` system variable. This variable has two possible values, which differ as follows:

- When `myisam_stats_method` is `nulls_equal`, all NULL values are treated as identical (that is, they all form a single value group).

If the NULL value group size is much higher than the average non-NULL value group size, this method skews the average value group size upward. This makes index appear to the optimizer to be less useful than it really is for joins that look for non-NULL values. Consequently, the `nulls_equal` method may cause the optimizer not to use the index for ref accesses when it should.

- When `myisam_stats_method` is `nulls_unequal`, NULL values are not considered the same. Instead, each NULL value forms a separate value group of size 1.

If you have many NULL values, this method skews the average value group size downward. If the average non-NULL value group size is large, counting NULL values each as a group of size 1 causes the optimizer to overestimate the value of the index for joins that look for non-NULL values. Consequently, the `nulls_unequal` method may cause the optimizer to use this index for ref lookups when other methods may be better.

If you tend to use many joins that use `<=>` rather than `=`, NULL values are not special in comparisons and one NULL is equal to another. In this case, `nulls_equal` is the appropriate statistics method.

The `myisam_stats_method` system variable has global and session values. Setting the global value affects MyISAM statistics collection for all MyISAM tables. Setting the session value affects statistics collection only for the current client connection. This means that you can force a table's statistics to be regenerated with a given method without affecting other clients by setting the session value of `myisam_stats_method`.

To regenerate table statistics, you can use any of the following methods:

- Set `myisam_stats_method`, and then issue a `CHECK TABLE` statement
- Execute **`myisamchk --stats_method=method_name --analyze`**
- Change the table to cause its statistics to go out of date (for example, insert a row and then delete it), and then set `myisam_stats_method` and issue an `ANALYZE TABLE` statement

Some caveats regarding the use of `myisam_stats_method`:

- You can force table statistics to be collected explicitly, as just described. However, MySQL may also collect statistics automatically. For example, if during the course of executing statements for a table, some of those statements modify the table, MySQL may collect statistics. (This may occur for bulk inserts or deletes, or some `ALTER TABLE` statements, for example.) If this happens, the statistics are collected using whatever value `myisam_stats_method` has at the time. Thus, if you collect statistics using one method, but `myisam_stats_method` is set to the other method when a table's statistics are collected automatically later, the other method will be used.
- There is no way to tell which method was used to generate statistics for a given MyISAM table.
- `myisam_stats_method` applies only to MyISAM tables. Other storage engines have only one method for collecting table statistics. Usually it is closer to the `nulls_equal` method.

7.4.8. How MySQL Opens and Closes Tables

When you execute a **`mysqladmin status`** command, you should see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The `Open tables` value of 12 can be somewhat puzzling if you have only six tables.

MySQL is multi-threaded, so there may be many clients issuing queries for a

given table simultaneously. To minimize the problem with multiple client threads having different states on the same table, the table is opened independently by each concurrent thread. This uses additional memory but normally increases performance. With MyISAM tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all threads.)

The `table_cache`, `max_connections`, and `max_tmp_tables` system variables affect the maximum number of files the server keeps open. If you increase one or more of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems allow you to increase the open-files limit, although the method varies widely from system to system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

`table_cache` is related to `max_connections`. For example, for 200 concurrent running connections, you should have a table cache size of at least $200 \times N$, where N is the maximum number of tables per join in any of the queries which you execute. You must also reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the `table_cache` setting. If `table_cache` is set too high, MySQL may run out of file descriptors and refuse connections, fail to perform queries, and be very unreliable. You also have to take into account that the MyISAM storage engine needs two file descriptors for each unique open table. You can increase the number of file descriptors available to MySQL using the `--open-files-limit` startup option to **mysqld**. See [Section A.2.17, “File Not Found”](#).

The cache of open tables is kept at a level of `table_cache` entries. The default value is 64; this can be changed with the `--table_cache` option to **mysqld**. Note that MySQL may temporarily open more tables than this to execute queries.

MySQL closes an unused table and removes it from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.

- When the cache contains more than `table_cache` entries and a table in the cache is no longer being used by any threads.
- When a table flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a **`mysqladmin flush-tables`** or **`mysqladmin refresh`** command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables that are not currently in use are released, beginning with the table least recently used.
- If a new table needs to be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary.

When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any MyISAM table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the `HANDLER tbl_name OPEN` statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls `HANDLER tbl_name CLOSE` or the thread terminates. When this happens, the table is put back in the table cache (if the cache is not full). See [Section 13.2.3, “HANDLER Syntax”](#).

You can determine whether your table cache is too small by checking the **`mysqld`** status variable `opened_tables`:

```
mysql> SHOW STATUS LIKE 'Opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 2741  |
```

+-----+-----+

If the value is very large, even when you have not issued many `FLUSH TABLES` statements, you should increase the table cache size. See [Section 5.2.2, “Server System Variables”](#), and [Section 5.2.4, “Server Status Variables”](#).

7.4.9. Drawbacks to Creating Many Tables in the Same Database

If you have many `MyISAM` tables in the same database directory, open, close, and create operations are slow. If you execute `SELECT` statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by making the table cache larger.

7.5. Optimizing the MySQL Server

7.5.1. System Factors and Startup Parameter Tuning

We start with system-level factors, because some of these decisions must be made very early to achieve large performance gains. In other cases, a quick look at this section may suffice. However, it is always nice to have a sense of how much can be gained by changing factors that apply at this level.

The operating system to use is very important. To get the best use of multiple-CPU machines, you should use Solaris (because its threads implementation works well) or Linux (because the 2.4 and later kernels have good SMP support). Note that older Linux kernels have a 2GB filesize limit by default. If you have such a kernel and a need for files larger than 2GB, you should get the Large File Support (LFS) patch for the ext2 filesystem. Other filesystems such as ReiserFS and XFS do not have this 2GB limitation.

Before using MySQL in production, we advise you to test it on your intended platform.

Other tips:

- If you have enough RAM, you could remove all swap devices. Some operating systems use a swap device in some contexts even if you have free memory.
- Avoid external locking. Since MySQL 4.0, the default has been for external locking to be disabled on all systems. The `--external-locking` and `--skip-external-locking` options explicitly enable and disable external locking.

Note that disabling external locking does not affect MySQL's functionality as long as you run only one server. Just remember to take down the server (or lock and flush the relevant tables) before you run **myisamchk**. On some systems it is mandatory to disable external locking because it does not work, anyway.

The only case in which you cannot disable external locking is when you run

multiple MySQL *servers* (not clients) on the same data, or if you run **myisamchk** to check (not repair) a table without telling the server to flush and lock the tables first. Note that using multiple MySQL servers to access the same data concurrently is generally *not* recommended, except when using MySQL Cluster.

The `LOCK TABLES` and `UNLOCK TABLES` statements use internal locking, so you can use them even if external locking is disabled.

7.5.2. Tuning Server Parameters

You can determine the default buffer sizes used by the **mysqld** server using this command:

```
shell> mysqld --verbose --help
```

This command produces a list of all **mysqld** options and configurable system variables. The output includes the default variable values and looks something like this:

```
back_log                50
binlog_cache_size       32768
bulk_insert_buffer_size 8388608
connect_timeout         5
date_format             (No default value)
datetime_format         (No default value)
default_week_format     0
delayed_insert_limit    100
delayed_insert_timeout  300
delayed_queue_size      1000
expire_logs_days        0
flush_time              1800
ft_max_word_len         84
ft_min_word_len         4
ft_query_expansion_limit 20
ft_stopword_file        (No default value)
group_concat_max_len    1024
innodb_additional_mem_pool_size 1048576
innodb_autoextend_increment 8
innodb_buffer_pool_awesome_mem_mb 0
innodb_buffer_pool_size 8388608
innodb_concurrency_tickets 500
innodb_file_io_threads  4
innodb_force_recovery   0
innodb_lock_wait_timeout 50
```

innodb_log_buffer_size	1048576
innodb_log_file_size	5242880
innodb_log_files_in_group	2
innodb_mirrored_log_groups	1
innodb_open_files	300
innodb_sync_spin_loops	20
innodb_thread_concurrency	8
innodb_thread_sleep_delay	10000
interactive_timeout	28800
join_buffer_size	131072
key_buffer_size	8388600
key_cache_age_threshold	300
key_cache_block_size	1024
key_cache_division_limit	100
long_query_time	10
lower_case_table_names	1
max_allowed_packet	1048576
max_binlog_cache_size	4294967295
max_binlog_size	1073741824
max_connect_errors	10
max_connections	100
max_delayed_threads	20
max_error_count	64
max_heap_table_size	16777216
max_join_size	4294967295
max_length_for_sort_data	1024
max_relay_log_size	0
max_seeks_for_key	4294967295
max_sort_length	1024
max_tmp_tables	32
max_user_connections	0
max_write_lock_count	4294967295
multi_range_count	256
myisam_block_size	1024
myisam_data_pointer_size	6
myisam_max_extra_sort_file_size	2147483648
myisam_max_sort_file_size	2147483647
myisam_repair_threads	1
myisam_sort_buffer_size	8388608
net_buffer_length	16384
net_read_timeout	30
net_retry_count	10
net_write_timeout	60
open_files_limit	0
optimizer_prune_level	1
optimizer_search_depth	62
preload_buffer_size	32768
query_alloc_block_size	8192
query_cache_limit	1048576
query_cache_min_res_unit	4096

query_cache_size	0
query_cache_type	1
query_cache_wlock_invalidate	FALSE
query_prealloc_size	8192
range_alloc_block_size	2048
read_buffer_size	131072
read_only	FALSE
read_rnd_buffer_size	262144
div_precision_increment	4
record_buffer	131072
relay_log_purge	TRUE
relay_log_space_limit	0
slave_compressed_protocol	FALSE
slave_net_timeout	3600
slave_transaction_retries	10
slow_launch_time	2
sort_buffer_size	2097144
sync-binlog	0
sync_frm	TRUE
sync-replication	0
sync-replication-slave-id	0
sync-replication-timeout	10
table_cache	64
thread_cache_size	0
thread_concurrency	10
thread_stack	196608
time_format	(No default value)
tmp_table_size	33554432
transaction_alloc_block_size	8192
transaction_prealloc_size	4096
updatable_views_with_limit	1
wait_timeout	28800

For a **mysqld** server that is currently running, you can see the current values of its system variables by connecting to it and issuing this statement:

```
mysql> SHOW VARIABLES;
```

You can also see some statistical and status indicators for a running server by issuing this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also can be obtained using **mysqladmin**:

```
shell> mysqladmin variables  
shell> mysqladmin extended-status
```

For a full description of all system and status variables, see [Section 5.2.2, “Server System Variables”](#), and [Section 5.2.4, “Server Status Variables”](#).

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally you get better performance by giving MySQL more memory.

When tuning a MySQL server, the two most important variables to configure are `key_buffer_size` and `table_cache`. You should first feel confident that you have these set appropriately before trying to change any other variables.

The following examples indicate some typical variable values for different runtime configurations.

- If you have at least 256MB of memory and many tables and want maximum performance with a moderate number of clients, you should use something like this:

```
shell> mysqld_safe --key_buffer_size=64M --table_cache=256 \  
                --sort_buffer_size=4M --read_buffer_size=1M &
```

- If you have only 128MB of memory and only a few tables, but you still do a lot of sorting, you can use something like this:

```
shell> mysqld_safe --key_buffer_size=16M --sort_buffer_size=1M
```

If there are very many simultaneous connections, swapping problems may occur unless **mysqld** has been configured to use very little memory for each connection. **mysqld** performs better if you have enough memory for all connections.

- With little memory and lots of connections, use something like this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \  
                --read_buffer_size=100K &
```

Or even this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \  
                --table_cache=32 --read_buffer_size=8K \  
                --net_buffer_length=1K &
```

If you are performing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, you should increase the value of `read_rnd_buffer_size` to speed up the reading of rows following sorting operations.

You can make use of the example option files included with your MySQL distribution; see [Section 4.3.2.1, “Preconfigured Option Files”](#).

If you specify an option on the command line for `mysqld` or `mysqld_safe`, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file.

To see the effects of a parameter change, do something like this:

```
shell> mysqld --key_buffer_size=32M --verbose --help
```

The variable values are listed near the end of the output. Make sure that the `--verbose` and `--help` options are last. Otherwise, the effect of any options listed after them on the command line are not reflected in the output.

For information on tuning the InnoDB storage engine, see [Section 14.2.11, “InnoDB Performance Tuning Tips”](#).

7.5.3. Controlling Query Optimizer Performance

The task of the query optimizer is to find an optimal plan for executing an SQL query. Because the difference in performance between “good” and “bad” plans can be orders of magnitude (that is, seconds versus hours or even days), most query optimizers, including that of MySQL, perform a more or less exhaustive search for an optimal plan among all possible query evaluation plans. For join queries, the number of possible plans investigated by the MySQL optimizer grows exponentially with the number of tables referenced in a query. For small numbers of tables (typically less than 7–10) this is not a problem. However, when larger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server's performance.

MySQL 5.0.1 introduces a more flexible method for query optimization that allows the user to control how exhaustive the optimizer is in its search for an optimal query evaluation plan. The general idea is that the fewer plans that are investigated by the optimizer, the less time it spends in compiling a query. On

the other hand, because the optimizer skips some plans, it may miss finding an optimal plan.

The behavior of the optimizer with respect to the number of plans it evaluates can be controlled via two system variables:

- The `optimizer_prune_level` variable tells the optimizer to skip certain plans based on estimates of the number of rows accessed for each table. Our experience shows that this kind of “educated guess” rarely misses optimal plans, and may dramatically reduce query compilation times. That is why this option is on (`optimizer_prune_level=1`) by default. However, if you believe that the optimizer missed a better query plan, this option can be switched off (`optimizer_prune_level=0`) with the risk that query compilation may take much longer. Note that, even with the use of this heuristic, the optimizer still explores a roughly exponential number of plans.
- The `optimizer_search_depth` variable tells how far into the “future” of each incomplete plan the optimizer should look to evaluate whether it should be expanded further. Smaller values of `optimizer_search_depth` may result in orders of magnitude smaller query compilation times. For example, queries with 12, 13, or more tables may easily require hours and even days to compile if `optimizer_search_depth` is close to the number of tables in the query. At the same time, if compiled with `optimizer_search_depth` equal to 3 or 4, the optimizer may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for `optimizer_search_depth`, this variable can be set to 0 to tell the optimizer to determine the value automatically.

7.5.4. How Compiling and Linking Affects the Speed of MySQL

Most of the following tests were performed on Linux with the MySQL benchmarks, but they should give some indication for other operating systems and workloads.

You obtain the fastest executables when you link with `-static`.

On Linux, it is best to compile the server with `pgcc` and `-O3`. You need about 200MB memory to compile `sql_yacc.cc` with these options, because `gcc` or

pgcc needs a great deal of memory to make all functions inline. You should also set `CXX=gcc` when configuring MySQL to avoid inclusion of the `libstdc++` library, which is not needed. Note that with some versions of **pgcc**, the resulting binary runs only on true Pentium processors, even if you use the compiler option indicating that you want the resulting code to work on all x586-type processors (such as AMD).

By using a better compiler and compilation options, you can obtain a 10–30% speed increase in applications. This is particularly important if you compile the MySQL server yourself.

When we tested both the Cygnus CodeFusion and Fujitsu compilers, neither was sufficiently bug-free to allow MySQL to be compiled with optimizations enabled.

The standard MySQL binary distributions are compiled with support for all character sets. When you compile MySQL yourself, you should include support only for the character sets that you are going to use. This is controlled by the `--with-charset` option to **configure**.

Here is a list of some measurements that we have made:

- If you use **pgcc** and compile everything with `-O6`, the **mysqld** server is 1% faster than with **gcc** 2.95.2.
- If you link dynamically (without `-static`), the result is 13% slower on Linux. Note that you still can use a dynamically linked MySQL library for your client applications. It is the server that is most critical for performance.
- For a connection from a client to a server running on the same host, if you connect using TCP/IP rather than a Unix socket file, performance is 7.5% slower. (On Unix, if you connect to the hostname `localhost`, MySQL uses a socket file by default.)
- For TCP/IP connections from a client to a server, connecting to a remote server on another host is 8–11% slower than connecting to a server on the same host, even for connections over 100Mb/s Ethernet.
- When running our benchmark tests using secure connections (all data encrypted with internal SSL support) performance was 55% slower than

with unencrypted connections.

- If you compile with `--with-debug=full`, most queries are 20% slower. Some queries may take substantially longer; for example, the MySQL benchmarks run 35% slower. If you use `--with-debug` (without `=full`), the speed decrease is only 15%. For a version of **mysqld** that has been compiled with `--with-debug=full`, you can disable memory checking at runtime by starting it with the `--skip-safemalloc` option. The execution speed should then be close to that obtained when configuring with `--with-debug`.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster than one compiled with **gcc** 3.2.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster in 32-bit mode than in 64-bit mode.
- Compiling with **gcc** 2.95.2 for UltraSPARC with the `-mcpu=v8 -wa, -xarch=v8plusa` options gives 4% more performance.
- On Solaris 2.5.1, MIT-pthreads is 8–12% slower than Solaris native threads on a single processor. With greater loads or more CPUs, the difference should be larger.
- Compiling on Linux-x86 using **gcc** without frame pointers (`-fomit-frame-pointer` or `-fomit-frame-pointer -ffixed-ebp`) makes **mysqld** 1–4% faster.

Binary MySQL distributions for Linux that are provided by MySQL AB used to be compiled with **pgcc**. We had to go back to regular **gcc** due to a bug in **pgcc** that would generate binaries that do not run on AMD. We will continue using **gcc** until that bug is resolved. In the meantime, if you have a non-AMD machine, you can build a faster binary by compiling with **pgcc**. The standard MySQL Linux binary is linked statically to make it faster and more portable.

7.5.5. How MySQL Uses Memory

The following list indicates some of the ways that the **mysqld** server uses memory. Where applicable, the name of the system variable relevant to the

memory use is given:

- The key buffer (variable `key_buffer_size`) is shared by all threads; other buffers used by the server are allocated as needed. See [Section 7.5.2, “Tuning Server Parameters”](#).
- Each connection uses some thread-specific space:
 - A stack (default 192KB, variable `thread_stack`)
 - A connection buffer (variable `net_buffer_length`)
 - A result buffer (variable `net_buffer_length`)

The connection buffer and result buffer are dynamically enlarged up to `max_allowed_packet` when needed. While a query is running, a copy of the current query string is also allocated.

- All threads share the same base memory.
- When a thread is no longer needed, the memory allocated to it is released and returned to the system unless the thread goes back into the thread cache. In that case, the memory remains allocated.
- Only compressed MyISAM tables are memory mapped. This is because the 32-bit memory space of 4GB is not large enough for most big tables. When systems with a 64-bit address space become more common, we may add general support for memory mapping.
- Each request that performs a sequential scan of a table allocates a *read buffer* (variable `read_buffer_size`).
- When reading rows in an arbitrary sequence (for example, following a sort), a *random-read buffer* (variable `read_rnd_buffer_size`) may be allocated in order to avoid disk seeks.
- All joins are executed in a single pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based hash tables. Temporary tables with a large row length (calculated as the sum of all column lengths) or that contain BLOB columns are stored on disk.

If an internal heap table exceeds the size of `tmp_table_size`, MySQL handles this automatically by changing the in-memory heap table to a disk-based MyISAM table as necessary. You can also increase the temporary table size by setting the `tmp_table_size` option to **mysqld**, or by setting the SQL option `SQL_BIG_TABLES` in the client program. See [Section 13.5.3, “SET Syntax”](#).

- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See [Section A.4.4, “Where MySQL Stores Temporary Files”](#).
- Almost all parsing and calculating is done in a local memory store. No memory overhead is needed for small items, so the normal slow memory allocation and freeing is avoided. Memory is allocated only for unexpectedly large strings. This is done with `malloc()` and `free()`.
- For each MyISAM table that is opened, the index file is opened once; the data file is opened once for each concurrently running thread. For each concurrent thread, a table structure, column structures for each column, and a buffer of size $3 \times N$ are allocated (where N is the maximum row length, not counting BLOB columns). A BLOB column requires five to eight bytes plus the length of the BLOB data. The MyISAM storage engine maintains one extra row buffer for internal use.
- For each table having BLOB columns, a buffer is enlarged dynamically to read in larger BLOB values. If you scan a table, a buffer as large as the largest BLOB value is allocated.
- Handler structures for all in-use tables are saved in a cache and managed as a FIFO. By default, the cache has 64 entries. If a table has been used by two running threads at the same time, the cache contains two entries for the table. See [Section 7.4.8, “How MySQL Opens and Closes Tables”](#).
- A `FLUSH TABLES` statement or **mysqldadmin flush-tables** command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. `FLUSH TABLES` does not return until all tables have been closed.

ps and other system status programs may report that **mysqld** uses a lot of

memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of **ps** counts the unused memory between stacks as used memory. You can verify this by checking available swap with `swap -s`. We test **mysqld** with several memory-leakage detectors (both commercial and Open Source), so there should be no memory leaks.

7.5.6. How MySQL Uses DNS

When a new client connects to **mysqld**, **mysqld** spawns a new thread to handle the request. This thread first checks whether the hostname is in the hostname cache. If not, the thread attempts to resolve the hostname:

- If the operating system supports the thread-safe `gethostbyaddr_r()` and `gethostbyname_r()` calls, the thread uses them to perform hostname resolution.
- If the operating system does not support the thread-safe calls, the thread locks a mutex and calls `gethostbyaddr()` and `gethostbyname()` instead. In this case, no other thread can resolve hostnames that are not in the hostname cache until the first thread unlocks the mutex.

You can disable DNS hostname lookups by starting **mysqld** with the `--skip-name-resolve` option. However, in this case, you can use only IP numbers in the MySQL grant tables.

If you have a very slow DNS and many hosts, you can get more performance by either disabling DNS lookups with `--skip-name-resolve` or by increasing the `HOST_CACHE_SIZE` define (default value: 128) and recompiling **mysqld**.

You can disable the hostname cache by starting the server with the `--skip-host-cache` option. To clear the hostname cache, issue a `FLUSH HOSTS` statement or execute the **mysqladmin flush-hosts** command.

To disallow TCP/IP connections entirely, start **mysqld** with the `--skip-networking` option.

7.6. Disk Issues

- Disk seeks are a huge performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.
- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:

- Using symbolic links

This means that, for MyISAM tables, you symlink the index file and data files from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read times better, assuming that the disk is not used for other purposes as well. See [Section 7.6.1, “Using Symbolic Links”](#).

- Striping

Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the N -th block on the $(N \text{ MOD } number_of_disks)$ disk, and so on. This means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See [Section 7.1.5, “Using Your Own Benchmarks”](#).

The speed difference for striping is *very* dependent on the parameters. Depending on how you set the striping parameters and number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.

- For reliability, you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need $2 \times N$ drives to hold N drives of data. This is probably

the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.

- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID N disk. RAID N can be a problem if you have many writes, due to the time required to update the parity bits.
- On Linux, you can get much more performance by using `hdparm` to configure your disk's interface. (Up to 100% under load is not uncommon.) The following `hdparm` options should be quite good for MySQL, and probably for many other applications:

```
hdparm -m 16 -d 1
```

Note that performance and reliability when using this command depend on your hardware, so we strongly suggest that you test your system thoroughly after using `hdparm`. Please consult the `hdparm` manual page for more information. If `hdparm` is not used wisely, filesystem corruption may result, so back up everything before experimenting!

- You can also set the parameters for the filesystem that the database uses:

If you do not need to know when files were last accessed (which is not really useful on a database server), you can mount your filesystems with the `-o noatime` option. That skips updates to the last access time in inodes on the filesystem, which avoids some disk seeks.

On many operating systems, you can set a filesystem to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you more performance without sacrificing too much reliability. (This flag is on by default on Linux.)

7.6.1. Using Symbolic Links

You can move tables and databases from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disk.

The recommended way to do this is simply to symlink databases to a different disk. Symlink tables only as a last resort.

7.6.1.1. Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is first to create a directory on some disk where you have free space and then to create a symlink to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
shell> ln -s /path/to/datadir
```

MySQL does not support linking one directory to multiple databases. Replacing a database directory with a symbolic link works as long as you do not make a symbolic link between databases. Suppose that you have a database db1 under the MySQL data directory, and then make a symlink db2 that points to db1:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

The result is that, or any table tb1_a in db1, there also appears to be a table tb1_a in db2. If one client updates db1.tb1_a and another client updates db2.tb1_a, problems are likely to occur.

However, if you really need to do this, it is possible by altering the source file `mysys/my_symlink.c`, in which you should look for the following statement:

```
if (!(MyFlags & MY_RESOLVE_LINK) ||
    (!lstat(filename, &stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

Change the statement to this:

```
if (1)
```

7.6.1.2. Using Symbolic Links for Tables on Unix

You should not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`). You can check whether your system supports symbolic links by issuing a `SHOW VARIABLES LIKE 'have_symlink'` statement.

Symlinks are fully supported only for MyISAM tables. For files used by tables for other storage engines, you may get strange problems if you try to use symbolic links.

The handling of symbolic links for MyISAM tables works as follows:

- In the data directory, you always have the table format (.frm) file, the data (.MYD) file, and the index (.MYI) file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The format file cannot.
- You can symlink the data file and the index file independently to different directories.
- You can instruct a running MySQL server to perform the symlinking by using the DATA DIRECTORY and INDEX DIRECTORY options to CREATE TABLE. See [Section 13.1.5, “CREATE TABLE Syntax”](#). Alternatively, symlinking can be accomplished manually from the command line using `ln -s` if **mysqld** is not running.
- **myisamchk** does not replace a symlink with the data file or index file. It works directly on the file to which the symlink points. Any temporary files are created in the directory where the data file or index file is located.
- **Note:** When you drop a table that is using symlinks, *both the symlink and the file to which the symlink points are dropped*. This is an extremely good reason why you should *not* run **mysqld** as the system root or allow system users to have write access to MySQL database directories.
- If you rename a table with `ALTER TABLE ... RENAME` and you do not move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you use `ALTER TABLE ... RENAME` to move a table to another database, the table is moved to the other database directory. The old symlinks and the files to which they pointed are deleted. In other words, the new table is not symlinked.
- If you are not using symlinks, you should use the `--skip-symbolic-links`

option to **mysqld** to ensure that no one can use **mysqld** to drop or rename a file outside of the data directory.

Table symlink operations that are not yet supported:

- ALTER TABLE ignores the DATA DIRECTORY and INDEX DIRECTORY table options.
- BACKUP TABLE and RESTORE TABLE do not respect symbolic links.
- The .frm file must *never* be a symbolic link (as indicated previously, only the data and index files can be symbolic links). Attempting to do this (for example, to make synonyms) produces incorrect results. Suppose that you have a database db1 under the MySQL data directory, a table tb11 in this database, and in the db1 directory you make a symlink tb12 that points to tb11:

```
shell> cd /path/to/datadir/db1
shell> ln -s tb11.frm tb12.frm
shell> ln -s tb11.MYD tb12.MYD
shell> ln -s tb11.MYI tb12.MYI
```

Problems result if one thread reads db1.tb11 and another thread updates db1.tb12:

- The query cache is “fooled” (it has no way of knowing that tb11 has not been updated, so it returns outdated results).
- ALTER statements on tb12 fail.

7.6.1.3. Using Symbolic Links for Databases on Windows

Symbolic links are enabled by default for all Windows servers. This enables you to put a database directory on a different disk by setting up a symbolic link to it. This is similar to the way that database symbolic links work on Unix, although the procedure for setting up the link is different. If you do not need symbolic links, you can disable them using the `--skip-symbolic-links` option.

On Windows, create a symbolic link to a MySQL database by creating a file in the data directory that contains the path to the destination directory. The file

should be named `db_name.sym`, where *db_name* is the database name.

Suppose that the MySQL data directory is `C:\mysql\data` and you want to have database `foo` located at `D:\data\foo`. Set up a symlink using this procedure

1. Make sure that the `D:\data\foo` directory exists by creating it if necessary. If you already have a database directory named `foo` in the data directory, you should move it to `D:\data`. Otherwise, the symbolic link will be ineffective. To avoid problems, make sure that the server is not running when you move the database directory.
2. Create a text file `C:\mysql\data\foo.sym` that contains the pathname `D:\data\foo\`.

After this, all tables created in the database `foo` are created in `D:\data\foo`. *Note that the symbolic link is not used if a directory with the same name as the database exists in the MySQL data directory.*

Chapter 8. Client and Utility Programs

Table of Contents

- [8.1. Overview of Client and Utility Programs](#)
- [8.2. **mysam_ftdump** — Display Full-Text Index information](#)
- [8.3. **mysamchk** — MyISAM Table-Maintenance Utility](#)
 - [8.3.1. **mysamchk** General Options](#)
 - [8.3.2. **mysamchk** Check Options](#)
 - [8.3.3. **mysamchk** Repair Options](#)
 - [8.3.4. Other **mysamchk** Options](#)
 - [8.3.5. **mysamchk** Memory Usage](#)
- [8.4. **mysamlog** — Display MyISAM Log File Contents](#)
- [8.5. **mysampack** — Generate Compressed, Read-Only MyISAM Tables](#)
- [8.6. **mysql** — The MySQL Command-Line Tool](#)
 - [8.6.1. **mysql** Options](#)
 - [8.6.2. **mysql** Commands](#)
 - [8.6.3. **mysql** Server-Side Help](#)
 - [8.6.4. Executing SQL Statements from a Text File](#)
 - [8.6.5. **mysql** Tips](#)
- [8.7. **mysql_explain_log** — Use EXPLAIN on Statements in Query Log](#)
- [8.8. **mysqlaccess** — Client for Checking Access Privileges](#)
- [8.9. **mysqladmin** — Client for Administering a MySQL Server](#)
- [8.10. **mysqlbinlog** — Utility for Processing Binary Log Files](#)
- [8.11. **mysqlcheck** — A Table Maintenance and Repair Program](#)
- [8.12. **mysqldump** — A Database Backup Program](#)
- [8.13. **mysqlhotcopy** — A Database Backup Program](#)
- [8.14. **mysqlimport** — A Data Import Program](#)
- [8.15. **mysqlshow** — Display Database, Table, and Column Information](#)
- [8.16. **mysql_zap** — Kill Processes That Match a Pattern](#)
- [8.17. **perror** — Explain Error Codes](#)
- [8.18. **replace** — A String-Replacement Utility](#)

There are many different MySQL client programs that connect to the server to access databases or perform administrative tasks. Other utilities are available as well. These do not establish a client connection with the server but perform MySQL-related operations.

This chapter provides a brief overview of these programs and then a more detailed description of each one. Each program's description indicates its invocation syntax and the options that it understands. See [Chapter 4, *Using MySQL Programs*](#), for general information on invoking programs and specifying program options.

8.1. Overview of Client and Utility Programs

The following list briefly describes the MySQL client programs and utilities:

- **myisam_ftdump**

A utility that displays information about full-text indexes in MyISAM tables. See [Section 8.2, “myisam_ftdump — Display Full-Text Index information”](#).

- **myisamchk**

A utility to describe, check, optimize, and repair MyISAM tables. See [Section 8.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

- **myisamlog**

A utility that processes the contents of a MyISAM log file. See [Section 8.4, “myisamlog — Display MyISAM Log File Contents”](#).

- **myisampack**

A utility that compresses MyISAM tables to produce smaller read-only tables. See [Section 8.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#).

- **mysql**

The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See [Section 8.6, “mysql — The MySQL Command-Line Tool”](#).

- **mysql_explain_log**

A utility that analyzes queries in the MySQL query log using EXPLAIN. See [Section 8.7, “mysql_explain_log — Use EXPLAIN on Statements in Query Log”](#).

- **mysqlaccess**

A script that checks the access privileges for a hostname, username, and database combination. See [Section 8.8, “mysqlaccess — Client for Checking Access Privileges”](#).

- **mysqladmin**

A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. **mysqladmin** can also be used to retrieve version, process, and status information from the server. See [Section 8.9, “mysqladmin — Client for Administering a MySQL Server”](#).

- **mysqlbinlog**

A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See [Section 8.10, “mysqlbinlog — Utility for Processing Binary Log Files”](#).

- **mysqlcheck**

A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See [Section 8.11, “mysqlcheck — A Table Maintenance and Repair Program”](#).

- **mysqldump**

A client that dumps a MySQL database into a file as SQL statements or as tab-separated text files. See [Section 8.12, “mysqldump — A Database Backup Program”](#).

- **mysqlhotcopy**

A utility that quickly makes backups of MyISAM tables while the server is running. See [Section 8.13, “mysqlhotcopy — A Database Backup Program”](#).

- **mysqlimport**

A client that imports text files into their respective tables using `LOAD DATA`

INFILE. See [Section 8.14, “mysqlimport — A Data Import Program”](#).

- **mysqlshow**

A client that displays information about databases, tables, columns, and indexes. See [Section 8.15, “mysqlshow — Display Database, Table, and Column Information”](#).

- **mysql_zap**

A utility that kills processes that match a pattern. [Section 8.16, “mysql_zap — Kill Processes That Match a Pattern”](#).

- **perror**

A utility that displays the meaning of system or MySQL error codes. See [Section 8.17, “perror — Explain Error Codes”](#).

- **replace**

A utility program that performs string replacement in the input text. See [Section 8.18, “replace — A String-Replacement Utility”](#).

MySQL AB also provides a number of GUI tools for administering and otherwise working with MySQL servers. For basic information about these, see [Chapter 4, Using MySQL Programs](#).

Each MySQL program takes many different options. Most programs provide a `-help` option that you can use to get a full description of the program's different options. For example, try **mysql --help**.

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables:

MYSQL_UNIX_PORT	The default Unix socket file; used for connections to localhost
MYSQL_TCP_PORT	The default port number; used for TCP/IP connections
MYSQL_PWD	The default password
MYSQL_DEBUG	Debug trace options when debugging

`TMPDIR` The directory where temporary tables and files are created

Use of `MYSQL_PWD` is insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

You can override the default option values or values specified in environment variables for all standard programs by specifying options in an option file or on the command line. See [Section 4.3, “Specifying Program Options”](#).

8.2. `myisam_ftdump` — Display Full-Text Index information

`myisam_ftdump` displays information about FULLTEXT indexes in MyISAM tables. It reads the MyISAM index file directly, so it must be run on the server host where the table is located

Invoke `myisam_ftdump` like this:

```
shell> myisam_ftdump [options] tbl_name index_num
```

The `tbl_name` argument should be the name of a MyISAM table. You can also specify a table by naming its index file (the file with the `.MYI` suffix). If you do not invoke `myisam_ftdump` in the directory where the table files are located, the table or index file name must be preceded by the pathname to the table's database directory. Index numbers begin with 0.

Example: Suppose that the test database contains a table named `mytexttable1` that has the following definition:

```
CREATE TABLE mytexttable
(
  id    INT NOT NULL,
  txt   TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
);
```

The index on `id` is index 0 and the FULLTEXT index on `txt` is index 1. If your working directory is the test database directory, invoke `myisam_ftdump` as follows:

```
shell> myisam_ftdump mytexttable 1
```

If the pathname to the test database directory is `/usr/local/mysql/data/test`, you can also specify the table name argument using that pathname. This is useful if you do not invoke `myisam_ftdump` in the database directory:

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

myisam_ftdump understands the following options:

- --help, -h -?

Display a help message and exit.

- --count, -c

Calculate per-word statistics (counts and global weights).

- --dump, -d

Dump the index, including data offsets and word weights.

- --length, -l

Report the length distribution.

- --stats, -s

Report global index statistics. This is the default operation if no other operation is specified.

- --verbose, -v

Verbose mode. Print more output about what the program does.

8.3. **myisamchk** — MyISAM Table-Maintenance Utility

The **myisamchk** utility gets information about your database tables or checks, repairs, or optimizes them. **myisamchk** works with MyISAM tables (tables that have `.MYD` and `.MYI` files for storing data and indexes).

Invoke **myisamchk** like this:

```
shell> myisamchk [options] tbl_name ...
```

The *options* specify what you want **myisamchk** to do. They are described in the following sections. You can also get a list of options by invoking **myisamchk --help**.

With no options, **myisamchk** simply checks your table as the default operation. To get more information or to tell **myisamchk** to take corrective action, specify options as described in the following discussion.

tbl_name is the database table you want to check or repair. If you run **myisamchk** somewhere other than in the database directory, you must specify the path to the database directory, because **myisamchk** has no idea where the database is located. In fact, **myisamchk** doesn't actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the **myisamchk** command line if you wish. You can also specify a table by naming its index file (the file with the `.MYI` suffix). This allows you to specify all tables in a directory by using the pattern `*.MYI`. For example, if you are in a database directory, you can check all the MyISAM tables in that directory like this:

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all MyISAM tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

If you want to check all MyISAM tables and repair any that are corrupted, you can use the following command:

```
shell> myisamchk --silent --force --fast --update-state \  
--key_buffer_size=64M --sort_buffer_size=64M \  
--read_buffer_size=1M --write_buffer_size=1M \  
/path/to/datadir/*/*.MYI
```

This command assumes that you have more than 64MB free. For more information about memory allocation with **myisamchk**, see [Section 8.3.5, “myisamchk Memory Usage”](#).

You must ensure that no other program is using the tables while you are running **myisamchk**. Otherwise, when you run **myisamchk**, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This means that you are trying to check a table that has been updated by another program (such as the **mysqld** server) that hasn't yet closed the file or that has died without closing the file properly.

If **mysqld** is running, you must force it to flush any table modifications that are still buffered in memory by using `FLUSH TABLES`. You should then ensure that no one is using the tables while you are running **myisamchk**. The easiest way to avoid this problem is to use `CHECK TABLE` instead of **myisamchk** to check tables.

8.3.1. myisamchk General Options

The options described in this section can be used for any type of table maintenance operation performed by **myisamchk**. The sections following this one describe options that pertain only to specific operations, such as table

checking or repairing.

- --help, -?

Display a help message and exit.

- --debug=debug_options, -# debug_options

Write a debugging log. The *debug_options* string often is 'd:t:o,file_name'.

- --silent, -s

Silent mode. Write output only when errors occur. You can use -s twice (-ss) to make **myisamchk** very silent.

- --verbose, -v

Verbose mode. Print more information about what the program does. This can be used with -d and -e. Use -v multiple times (-vv, -vvv) for even more output.

- --version, -V

Display version information and exit.

- --wait, -w

Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. Note that if you are running **mysqld** with external locking disabled, the table can be locked only by another **myisamchk** command.

You can also set the following variables by using --var_name=value syntax:

Variable	Default Value
decode_bits	9
ft_max_word_len	version-dependent
ft_min_word_len	4

ft_stopword_file	built-in list
key_buffer_size	523264
myisam_block_size	1024
read_buffer_size	262136
sort_buffer_size	2097144
sort_key_blocks	16
stats_method	nulls_unequal
write_buffer_size	262136

The possible **myisamchk** variables and their default values can be examined with **myisamchk --help**:

`sort_buffer_size` is used when the keys are repaired by sorting keys, which is the normal case when you use `--recover`.

`key_buffer_size` is used when you are checking the table with `--extend-check` or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use `--safe-recover`.
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for CHAR, VARCHAR, or TEXT columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force **myisamchk** to repair by sorting, you can use the `--sort-recover` option.

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the `key_buffer_size` and `sort_buffer_size` variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

`myisam_block_size` is the size used for index blocks.

`stats_method` influences how NULL values are treated for index statistics collection when the `--analyze` option is given. It acts like the `myisam_stats_method` system variable. For more information, see the description of `myisam_stats_method` in [Section 5.2.2, “Server System Variables”](#), and [Section 7.4.7, “MyISAM Index Statistics Collection”](#). For MySQL 5.0, `stats_method` was added in MySQL 5.0.14. For older versions, the statistics collection method is equivalent to `nulls_equal`.

`ft_min_word_len` and `ft_max_word_len` indicate the minimum and maximum word length for FULLTEXT indexes. `ft_stopword_file` names the stopword file. These need to be set under the following circumstances.

If you use **myisamchk** to perform an operation that modifies table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values to **myisamchk** that you use for **mysqld**. For example, if you have set the minimum word length to 3, you can repair a table with **myisamchk** like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that **myisamchk** and the server use the same values for full-text parameters, you can place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3
```

```
[myisamchk]
ft_min_word_len=3
```

An alternative to using **myisamchk** is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE`. These statements are performed by the server, which knows the proper full-text parameter values to use.

8.3.2. **myisamchk** Check Options

myisamchk supports the following options for table checking operations:

- `--check, -c`

Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- `--check-only-changed, -C`

Check only tables that have changed since the last check.

- `--extend-check, -e`

Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, **myisamchk** or **myisamchk --medium-check** should be able to determine whether there are any errors in the table.

If you are using `--extend-check` and have plenty of memory, setting the `key_buffer_size` variable to a large value helps the repair operation run faster.

- `--fast, -F`

Check only tables that haven't been closed properly.

- `--force, -f`

Do a repair operation automatically if **myisamchk** finds any errors in the table. The repair type is the same as that specified with the `--recover` or `-r` option.

- `--information, -i`

Print informational statistics about the table that is checked.

- `--medium-check, -m`

Do a check that is faster than an `--extend-check` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--read-only, -T`

Don't mark the table as checked. This is useful if you use **myisamchk** to check a table that is in use by some other application that doesn't use locking, such as **mysqld** when run with external locking disabled.

- `--update-state, -U`

Store information in the `.MYI` file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the `--check-only-changed` option, but you shouldn't use this option if the **mysqld** server is using the table and you are running it with external locking disabled.

8.3.3. myisamchk Repair Options

myisamchk supports the following options for table repair operations:

- `--backup, -B`

Make a backup of the `.MYD` file as `file_name-time.BAK`

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--correct-checksum`

Correct the checksum information for the table.

- `--data-file-length=len, -D len`

Maximum length of the data file (when re-creating data file when it is “full”).

- `--extend-check, -e`

Do a repair that tries to recover every possible row from the data file. Normally, this also finds a lot of garbage rows. Don't use this option unless you are desperate.

- `--force, -f`

Overwrite old intermediate files (files with names like `tbl_name.TMD`) instead of aborting.

- `--keys-used=val, -k val`

For **myisamchk**, the option value is a bit-value that indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. An option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using **myisamchk -r**.

- `--max-record-length=len`

Skip rows larger than the given length if **myisamchk** cannot allocate memory to hold them.

- `--parallel-recover, -p`

Uses the same technique as `-r` and `-n`, but creates all the keys in parallel, using different threads. *This is beta-quality code. Use at your own risk!*

- `--quick, -q`

Achieve a faster repair by not modifying the data file. You can specify this option twice to force **myisamchk** to modify the original data file in case of duplicate keys.

- `--recover, -r`

Do a repair that can fix almost any problem except unique keys that aren't unique (which is an extremely unlikely error with MyISAM tables). If you want to recover a table, this is the option to try first. You should try `--safe-recover` only if **myisamchk** reports that the table can't be recovered using `-recover`. (In the unlikely case that `--recover` fails, the data file remains

intact.)

If you have lots of memory, you should increase the value of `sort_buffer_size`.

- `--safe-recover, -o`

Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than `--recover`, but can handle a couple of very unlikely cases that `--recover` cannot. This recovery method also uses much less disk space than `--recover`. Normally, you should repair first with `--recover`, and then with `--safe-recover` only if `--recover` fails.

If you have lots of memory, you should increase the value of `key_buffer_size`.

- `--set-character-set=name`

Change the character set used by the table indexes. This option was replaced by `--set-collation` in MySQL 5.0.3.

- `--set-collation=name`

Specify the collation to use for sorting table indexes. The character set name is implied by the first part of the collation name. This option was added in MySQL 5.0.3.

- `--sort-recover, -n`

Force **myisamchk** to use sorting to resolve the keys even if the temporary files would be very large.

- `--tmpdir=path, -t path`

Path of the directory to be used for storing temporary files. If this is not set, **myisamchk** uses the value of the `TMPDIR` environment variable. `tmpdir` can be set to a list of directory paths that are used successively in round-robin fashion for creating temporary files. The separator character between directory names is the colon (':') on Unix and the semicolon(';') on

Windows, NetWare, and OS/2.

- --unpack, -u

Unpack a table that was packed with **mysampack**.

8.3.4. Other **myisamchk** Options

myisamchk supports the following options for actions other than table checks and repairs:

- --analyze, -a

Analyze the distribution of key values. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use. To obtain information about the key distribution, use a **myisamchk --description --verbose *tbl_name*** command or the `SHOW INDEX FROM tbl_name` statement.

- --block-search=offset, -b offset

Find the record that a block at the given offset belongs to.

- --description, -d

Print some descriptive information about the table.

- --set-auto-increment[=value], -A[value]

Force `AUTO_INCREMENT` numbering for new records to start at the given value (or higher, if there are existing records with `AUTO_INCREMENT` values this large). If *value* is not specified, `AUTO_INCREMENT` numbers for new records begin with the largest value currently in the table, plus one.

- --sort-index, -S

Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.

- --sort-records=N, -R N

Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW INDEX`, which displays a table's indexes in the same order that **myisamchk** sees them. Indexes are numbered beginning with 1.

If keys are not packed (`PACK_KEYS=0`), they have the same length, so when **myisamchk** sorts and moves records, it just overwrites record offsets in the index. If keys are packed (`PACK_KEYS=1`), **myisamchk** must unpack key blocks first, then re-create indexes and pack the key blocks again. (In this case, re-creating indexes is faster than updating offsets for each index.)

8.3.5. myisamchk Memory Usage

Memory allocation is important when you run **myisamchk**. **myisamchk** uses no more memory than its memory-related variables are set to. If you are going to use **myisamchk** on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get **myisamchk** to operate faster. For example, if you have more than 32MB RAM, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk --sort_buffer_size=16M --key_buffer_size=16M \  
             --read_buffer_size=1M --write_buffer_size=1M ...
```

Using `--sort_buffer_size=16M` should probably be enough for most cases.

Be aware that **myisamchk** uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory filesystem, you may easily get out of memory errors. If this happens, run **myisamchk** with the `--tmpdir=path` option to specify some directory located on a filesystem that has more space.

When repairing, **myisamchk** also needs a lot of disk space:

- Double the size of the data file (the original file and a copy). This space is not needed if you do a repair with `--quick`; in this case, only the index file is re-created. This space is needed on the same filesystem as the original data file! (The copy is created in the same directory as the original.)

- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space is needed on the same filesystem as the original index file!
- When using `--recover` or `--sort-recover` (but not when using `--safe-recover`), you need space for a sort buffer. The following formula yields the amount of space required:

$$(largest_key + row_pointer_length) \times number_of_rows \times 2$$

You can check the length of the keys and the `row_pointer_length` with **`myisamchk -dv tbl_name`**. This space is allocated in the temporary directory (specified by `TMPDIR` or `--tmpdir=path`).

If you have a problem with disk space during repair, you can try `--safe-recover` instead of `--recover`.

8.4. **myisamlog** — Display MyISAM Log File Contents

myisamlog processes the contents of a MyISAM log file.

Invoke **myisamlog** like this:

```
shell> myisamlog [options] [log_file [tbl_name] ...]
```

The default operation is update (-u). If a recovery is done (-r), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is `myisam.log` if no *log_file* argument is given, If tables are named on the command line, only those tables are updated.

myisamlog understands the following options:

- -?, -I

Display a help message and exit.

- -c N

Execute only *N* commands.

- -f N

Specify the maximum number of open files.

- -i

Display extra information before exiting.

- -o offset

Specify the starting offset.

- -p N

Remove *N* components from path.

- -r

Perform a recovery operation.

- -R record_pos_file record_pos

Specify record position file and record position.

- -u

Perform an update operation.

- -v

Verbose mode. Print more output about what the program does. This option can be given multiple times to produce more and more output.

- -w write_file

Specify the write file.

- -V

Display version information.

8.5. **myisampack** — Generate Compressed, Read-Only MyISAM Tables

The **myisampack** utility compresses MyISAM tables. **myisampack** works by compressing each column in the table separately. Usually, **myisampack** packs the data file 40%-70%.

When the table is used later, the server reads into memory the information needed to decompress columns. This results in much better performance when accessing individual rows, because you only have to uncompress exactly one row.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` does not work, MySQL falls back to normal read/write file operations.

Please note the following:

- If the **mysqld** server was invoked with external locking disabled, it is not a good idea to invoke **myisampack** if the table might be updated by the server during the packing process. It is safest to compress tables with the server stopped.
- After packing a table, it becomes read-only. This is generally intended (such as when accessing packed tables on a CD). Allowing writes to a packed table is on our TODO list, but with low priority.
- **myisampack** can pack BLOB or TEXT columns. (The older **pack_isam** program for ISAM tables did not have this capability.)

Invoke **myisampack** like this:

```
shell> myisampack [options] file_name ...
```

Each filename argument should be the name of an index (.MYI) file. If you are not in the database directory, you should specify the pathname to the file. It is permissible to omit the .MYI extension.

After you compress a table with **myisampack**, you should use **myisamchk -rq** to rebuild its indexes. [Section 8.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

myisampack supports the following options:

- `--help, -?`

Display a help message and exit.

- `--backup, -b`

Make a backup of each table's data file using the name `tbl_name.OLD`.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The *debug_options* string often is `'d:t:o,file_name'`.

- `--force, -f`

Produce a packed table even if it becomes larger than the original or if the intermediate file from an earlier invocation of **myisampack** exists.

(**myisampack** creates an intermediate file named `tbl_name.TMD` in the database directory while it compresses the table. If you kill **myisampack**, the `.TMD` file might not be deleted.) Normally, **myisampack** exits with an error if it finds that `tbl_name.TMD` exists. With `--force`, **myisampack** packs the table anyway.

- `--join=big_tbl_name, -j big_tbl_name`

Join all tables named on the command line into a single table *big_tbl_name*. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

- `--packlength=len, -p len`

Specify the row length storage size, in bytes. The value should be 1, 2, or 3. **myisampack** stores all rows with length pointers of 1, 2, or 3 bytes. In most normal cases, **myisampack** can determine the correct length value before it begins packing the file, but it may notice during the packing process that it could have used a shorter length. In this case, **myisampack** prints a note that you could use a shorter row length the next time you pack the same file.

- `--silent, -s`

Silent mode. Write output only when errors occur.

- `--test, -t`

Do not actually pack the table, just test packing it.

- `--tmpdir=path, -T path`

Use the named directory as the location where **myisampack** creates temporary files.

- `--verbose, -v`

Verbose mode. Write information about the progress of the packing operation and its result.

- `--version, -V`

Display version information and exit.

- `--wait, -w`

Wait and retry if the table is in use. If the **mysqld** server was invoked with external locking disabled, it is not a good idea to invoke **myisampack** if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```

shell> ls -l station.*
-rw-rw-r--  1 monty    my          994128 Apr 17 19:00 station.MYD
-rw-rw-r--  1 monty    my          53248  Apr 17 19:00 station.MYI
-rw-rw-r--  1 monty    my           5767 Apr 17 19:00 station.frm

```

```

shell> myisamchk -dvv station

```

```

MyISAM file:      station
Isam-version:    2
Creation time:   1996-03-13 10:08:58
Recover time:   1997-02-02  3:06:43
Data records:   1192 Deleted blocks:      0
Datafile parts: 1192 Deleted data:        0
Datafile pointer (bytes): 2 Keyfile pointer (bytes): 2
Max datafile length: 54657023 Max keyfile length: 33554431
Recordlength:   834
Record format:  Fixed length

```

table description:

Key	Start	Len	Index	Type	Root	Blocksize	Rec/ke
1	2	4	unique	unsigned long	1024	1024	
2	32	30	multip.	text	10240	1024	

Field Start Length Type

1	1	1	
2	2	4	
3	6	4	
4	10	1	
5	11	20	
6	31	1	
7	32	30	
8	62	35	
9	97	35	
10	132	35	
11	167	4	
12	171	16	
13	187	35	
14	222	4	
15	226	16	
16	242	20	
17	262	20	
18	282	20	
19	302	30	
20	332	4	
21	336	4	
22	340	1	
23	341	8	
24	349	8	
25	357	8	
26	365	2	

27	367	2
28	369	4
29	373	4
30	377	1
31	378	2
32	380	8
33	388	4
34	392	4
35	396	4
36	400	4
37	404	1
38	405	4
39	409	4
40	413	4
41	417	4
42	421	4
43	425	4
44	429	20
45	449	30
46	479	1
47	480	1
48	481	79
49	560	79
50	639	79
51	718	79
52	797	8
53	805	1
54	806	1
55	807	20
56	827	4
57	831	4

```
shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics
```

```
normal:      20  empty-space:   16  empty-zero:    12  empty-fill:
pre-space:   0   end-space:    12  table-lookups: 5   zero:
Original trees: 57  After join: 17
```

```
- Compressing file
```

```
87.14%
```

```
Remember to run myisamchk -rq on compressed tables
```

```
shell> ls -l station.*
```

```
-rw-rw-r--  1 monty  my      127874 Apr 17 19:00 station.MYD
-rw-rw-r--  1 monty  my      55296 Apr 17 19:04 station.MYI
-rw-rw-r--  1 monty  my       5767 Apr 17 19:00 station.frm
```

```
shell> myisamchk -dvv station
```

MyISAM file: station
 Isam-version: 2
 Creation time: 1996-03-13 10:08:58
 Recover time: 1997-04-17 19:04:26
 Data records: 1192 Deleted blocks: 0
 Datafile parts: 1192 Deleted data: 0
 Datafile pointer (bytes): 3 Keyfile pointer (bytes): 1
 Max datafile length: 16777215 Max keyfile length: 131071
 Recordlength: 834
 Record format: Compressed

table description:

Key	Start	Len	Index	Type	Root	Blocksize	Rec/ke
1	2	4	unique	unsigned long	10240	1024	
2	32	30	multip.	text	54272	1024	

Field	Start	Length	Type	Huff tree	Bits
1	1	1	constant	1	0
2	2	4	zerofill(1)	2	9
3	6	4	no zeros, zerofill(1)	2	9
4	10	1		3	9
5	11	20	table-lookup	4	0
6	31	1		3	9
7	32	30	no endspace, not_always	5	9
8	62	35	no endspace, not_always, no empty	6	9
9	97	35	no empty	7	9
10	132	35	no endspace, not_always, no empty	6	9
11	167	4	zerofill(1)	2	9
12	171	16	no endspace, not_always, no empty	5	9
13	187	35	no endspace, not_always, no empty	6	9
14	222	4	zerofill(1)	2	9
15	226	16	no endspace, not_always, no empty	5	9
16	242	20	no endspace, not_always	8	9
17	262	20	no endspace, no empty	8	9
18	282	20	no endspace, no empty	5	9
19	302	30	no endspace, no empty	6	9
20	332	4	always zero	2	9
21	336	4	always zero	2	9
22	340	1		3	9
23	341	8	table-lookup	9	0
24	349	8	table-lookup	10	0
25	357	8	always zero	2	9
26	365	2		2	9
27	367	2	no zeros, zerofill(1)	2	9
28	369	4	no zeros, zerofill(1)	2	9
29	373	4	table-lookup	11	0
30	377	1		3	9
31	378	2	no zeros, zerofill(1)	2	9
32	380	8	no zeros	2	9
33	388	4	always zero	2	9

34	392	4	table-lookup	12	0
35	396	4	no zeros, zerofill(1)	13	9
36	400	4	no zeros, zerofill(1)	2	9
37	404	1		2	9
38	405	4	no zeros	2	9
39	409	4	always zero	2	9
40	413	4	no zeros	2	9
41	417	4	always zero	2	9
42	421	4	no zeros	2	9
43	425	4	always zero	2	9
44	429	20	no empty	3	9
45	449	30	no empty	3	9
46	479	1		14	4
47	480	1		14	4
48	481	79	no endspace, no empty	15	9
49	560	79	no empty	2	9
50	639	79	no empty	2	9
51	718	79	no endspace	16	9
52	797	8	no empty	2	9
53	805	1		17	1
54	806	1		3	9
55	807	20	no empty	3	9
56	827	4	no zeros, zerofill(2)	2	9
57	831	4	no zeros, zerofill(1)	2	9

mysampack displays the following kinds of information:

- normal

The number of columns for which no extra packing is used.

- empty-space

The number of columns containing values that are only spaces. These occupy one bit.

- empty-zero

The number of columns containing values that are only binary zeros. These occupy one bit.

- empty-fill

The number of integer columns that do not occupy the full byte range of their type. These are changed to a smaller type. For example, a `BIGINT`

column (eight bytes) can be stored as a TINYINT column (one byte) if all its values are in the range from -128 to 127.

- pre-space

The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- end-space

The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- table-lookup

The column had only a small number of different values, which were converted to an ENUM before Huffman compression.

- zero

The number of columns for which all values are zero.

- Original trees

The initial number of Huffman trees.

- After join

The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, **myisamchk -dvv** prints additional information about each column:

- Type

The data type. The value may contain any of the following descriptors:

- constant

All rows have the same value.

- no endspace

Do not store endspace.

- no endspace, not_always

Do not store endspace and do not do endspace compression for all values.

- no endspace, no empty

Do not store endspace. Do not store empty values.

- table-lookup

The column was converted to an ENUM.

- zerofill(N)

The most significant N bytes in the value are always 0 and are not stored.

- no zeros

Do not store zeros.

- always zero

Zero values are stored using one bit.

- Huff tree

The number of the Huffman tree associated with the column.

- Bits

The number of bits used in the Huffman tree.

After you run **myisampack**, you must run **myisamchk** to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

After you have installed the packed table into the MySQL database directory, you should execute **mysqladmin flush-tables** to force **mysqld** to start using the new table.

To unpack a packed table, use the **--unpack** option to **myisamchk**.

8.6. `mysql` — The MySQL Command-Line Tool

`mysql` is a simple SQL shell (with GNU `readline` capabilities). It supports interactive and non-interactive use. When used interactively, query results are presented in an ASCII-table format. When used non-interactively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the `--quick` option. This forces `mysql` to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the `mysql_use_result()` C API function in the client/server library rather than `mysql_store_result()`.

Using `mysql` is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password=your_password db_name
```

Then type an SQL statement, end it with `;`, `\g`, or `\G` and press Enter.

You can execute SQL statements in a script file (batch file) like this:

```
shell> mysql db_name < script.sql > output.tab
```

8.6.1. `mysql` Options

`mysql` supports the following options:

- `--help, -?`

Display a help message and exit.

- `--auto-rehash`

Enable automatic rehashing. This option is on by default, which enables table and column name completion. Use `--skip-auto-rehash` to disable rehashing. That causes **mysql** to start faster, but you must issue the rehash command if you want to use table and column name completion.

- `--batch, -B`

Print results using tab as the column separator, with each row on a new line. With this option, **mysql** does not use the history file.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--column-names`

Write column names in results.

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--database=db_name, -D db_name`

The database to use. This is useful primarily in an option file.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The *debug_options* string often is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysql.trace'`.

- `--debug-info, -T`

Print some debugging information when the program exits.

- `--default-character-set=charset_name`

Use *charset_name* as the default character set. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--delimiter=str`

Set the statement delimiter. The default is the semicolon character (`;`).

- `--execute=statement, -e statement`

Execute the statement and quit. The default output format is like that produced with `--batch`. See [Section 4.3.1, “Using Options on the Command Line”](#), for some examples.

- `--force, -f`

Continue even if an SQL error occurs.

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--html, -H`

Produce HTML output.

- `--ignore-spaces, -i`

Ignore spaces after function names. The effect of this is described in the discussion for the `IGNORE_SPACE` SQL mode (see [Section 5.2.5, “The Server SQL Mode”](#)).

- `--line-numbers`

Write line numbers for errors. Disable this with `--skip-line-numbers`.

- `--local-infile[={0|1}]`

Enable or disable `LOCAL` capability for `LOAD DATA INFILE`. With no value, the option enables `LOCAL`. The option may be given as `--local-infile=0` or `--local-infile=1` to explicitly disable or enable `LOCAL`. Enabling `LOCAL` has no effect if the server does not also support it.

- `--named-commands, -G`

Enable named **mysql** commands. Long-format commands are allowed, not just short-format commands. For example, `quit` and `\q` both are recognized. Use `--skip-named-commands` to disable named commands. See [Section 8.6.2, “mysql Commands”](#).

- `--no-auto-rehash, -A`

Deprecated form of `-skip-auto-rehash`. See the description for `--auto-rehash`.

- `--no-beep, -b`

Do not beep when errors occur.

- `--no-named-commands, -g`

Disable named commands. Use the `*` form only, or use named commands only at the beginning of a line ending with a semicolon (`;`). **mysql** starts with this option *enabled* by default. However, even with this option, long-format commands still work from the first line. See [Section 8.6.2, “mysql Commands”](#).

- `--no-pager`

Deprecated form of `--skip-pager`. See the `--pager` option.

- `--no-tee`

Do not copy output to a file. [Section 8.6.2, “mysql Commands”](#), discusses tee files further.

- `--one-database, -o`

Ignore statements except those for the default database named on the command line. This is useful for skipping updates to other databases in the binary log.

- `--pager [=command]`

Use the given command for paging query output. If the command is

omitted, the default pager is the value of your `PAGER` environment variable. Valid pagers are **less**, **more**, **cat** [**> filename**], and so forth. This option works only on Unix. It does not work in batch mode. To disable paging, use `--skip-pager`. [Section 8.6.2, “mysql Commands”](#), discusses output paging further.

- `--password[=password]`, `-p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- `--port=port_num`, `-P port_num`

The TCP/IP port number to use for the connection.

- `--prompt=format_str`

Set the prompt to the specified format. The default is `mysql>`. The special sequences that the prompt can contain are described in [Section 8.6.2, “mysql Commands”](#).

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--quick`, `-q`

Do not cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, **mysql** does not use the history file.

- `--raw`, `-r`

Write column values without escape conversion. Often used with the `--batch` option.

- `--reconnect`

If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use `--skip-reconnect`.

- `--safe-updates`, `--i-am-a-dummy`, `-U`

Allow only those `UPDATE` and `DELETE` statements that specify which rows to modify by using key values. If you have set this option in an option file, you can override it by using `--safe-updates` on the command line. See [Section 8.6.5, “mysql Tips”](#), for more information about this option.

- `--secure-auth`

Do not send passwords to the server in old (pre-4.1.1) format. This prevents connections except for servers that use the newer password format.

- `--show-warnings`

Cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode. This option was added in MySQL 5.0.6.

- `--sigint-ignore`

Ignore `SIGINT` signals (typically the result of typing Control-C).

- `--silent`, `-s`

Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

- `--skip-column-names`, `-N`

Do not write column names in results.

- `--skip-line-numbers`, `-L`

Do not write line numbers for errors. Useful when you want to compare result files that include error messages.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.9.7.3, “SSL Command Options”](#).

- `--table, -t`

Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- `--tee=file_name`

Append a copy of output to the given file. This option does not work in batch mode. in [Section 8.6.2, “mysql Commands”](#), discusses tee files further.

- `--unbuffered, -n`

Flush the buffer after each query.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Produce more output about what the program does. This option can be given multiple times to produce more and more output. (For example, `-v -v -v` produces table output format even in batch mode.)

- `--version, -V`

Display version information and exit.

- `--vertical, -E`

Print query output rows vertically (one line per column value). Without this option, you can specify vertical output for individual statements by terminating them with \G.

- `--wait, -w`

If the connection cannot be established, wait and retry instead of aborting.

- `--xml, -X`

Produce XML output.

You can also set the following variables by using `--var_name=value` syntax:

- `connect_timeout`

The number of seconds before connection timeout. (Default value is 0.)

- `max_allowed_packet`

The maximum packet length to send to or receive from the server. (Default value is 16MB.)

- `max_join_size`

The automatic limit for rows in a join when using `--safe-updates`. (Default value is 1,000,000.)

- `net_buffer_length`

The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `select_limit`

The automatic limit for `SELECT` statements when using `--safe-updates`. (Default value is 1,000.)

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

On Unix, the **mysql** client writes a record of executed statements to a history file. By default, the history file is named `.mysql_history` and is created in your home directory. To specify a different file, set the value of the `MYSQL_HISTFILE` environment variable.

If you do not want to maintain a history file, first remove `.mysql_history` if it exists, and then use either of the following techniques:

- Set the `MYSQL_HISTFILE` variable to `/dev/null`. To cause this setting to take effect each time you log in, put the setting in one of your shell's startup files.
- Create `.mysql_history` as a symbolic link to `/dev/null`:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

You need do this only once.

8.6.2. mysql Commands

mysql sends each SQL statement that you issue to the server to be executed. There is also a set of commands that **mysql** itself interprets. For a list of these commands, type `help` or `\h` at the `mysql>` prompt:

```
mysql> help
```

List of all MySQL commands:

Note that all text commands must be first on line and end with ';'.

```
?          (\?) Synonym for `help'.
clear      (\c) Clear command.
connect    (\r) Reconnect to the server. Optional arguments are db an
delimiter (\d) Set statement delimiter. NOTE: Takes the rest of the
           new delimiter.
edit       (\e) Edit command with $EDITOR.
ego        (\G) Send command to mysql server, display result vertical
exit       (\q) Exit mysql. Same as quit.
go         (\g) Send command to mysql server.
help       (\h) Display this help.
nopager    (\n) Disable pager, print to stdout.
notee      (\t) Don't write into outfile.
pager      (\P) Set PAGER [to_pager]. Print the query results via PAG
print      (\p) Print current command.
prompt     (\R) Change your mysql prompt.
quit       (\q) Quit mysql.
```

```

rehash      (\#) Rebuild completion hash.
source      (\.) Execute an SQL script file. Takes a file name as an a
status      (\s) Get status information from the server.
system      (\!) Execute a system shell command.
tee         (\T) Set outfile [to_outfile]. Append everything into give
            outfile.
use         (\u) Use another database. Takes database name as argument
charset     (\C) Switch to another charset. Might be needed for proces
warnings    (\W) Show warnings after every statement.
nowarning   (\w) Don't show warnings after every statement.

```

For server side help, type 'help contents'

Each command has both a long and short form. The long form is not case sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

If you provide an argument to the help command, **mysql** uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. For more information, see [Section 8.6.3, “mysql Server-Side Help”](#).

In the `delimiter` command, you should avoid the use of the backslash (‘\’) character because that is the escape character for MySQL.

The **edit**, **nopager**, **pager**, and **system** commands work only in Unix.

The `status` command provides some information about the connection and the server you are using. If you are running in `--safe-updates` mode, `status` also prints the values for the **mysql** variables that affect your queries.

To log queries and their output, use the **tee** command. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. You can enable this feature on the command line with the `--tee` option, or interactively with the **tee** command. The **tee** file can be disabled interactively with the **notee** command. Executing **tee** again re-enables logging. Without a parameter, the previous file is used. Note that **tee** flushes query results to the file after each statement, just before **mysql** prints its next prompt.

By using the `--pager` option, it is possible to browse or search query results in interactive mode with Unix programs such as **less**, **more**, or any other similar program. If you specify no value for the option, **mysql** checks the value of the `PAGER` environment variable and sets the pager to that. Output paging can be

enabled interactively with the **pager** command and disabled with **nopager**. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or `stdout` if no pager was specified.

Output paging works only in Unix because it uses the `popen()` function, which does not exist on Windows. For Windows, the **tee** option can be used instead to save query output, although this is not as convenient as **pager** for browsing output in some situations.

Here are a few tips about the **pager** command:

- You can use it to write to a file and the results go only to the file:

```
mysql> pager cat > /tmp/log.txt
```

You can also pass any options for the program that you want to use as your pager:

```
mysql> pager less -n -i -S
```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to **less** can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within **less** to switch the horizontal-browse mode on and off. For more information, read the **less** manual page:

```
shell> man less
```

- You can specify very complex pager commands for handling query output:

```
mysql> pager cat | tee /dr1/tmp/res.txt \  
          | tee /dr2/tmp/res2.txt | less -n -i -S
```

In this example, the command would send query results to two files in two different directories on two different filesystems mounted on `/dr1` and `/dr2`, yet still display the results onscreen via **less**.

You can also combine the **tee** and **pager** functions. Have a **tee** file enabled and

pager set to **less**, and you are able to browse the results using the **less** program and still have everything appended into a file the same time. The difference between the Unix **tee** used with the **pager** command and the **mysql** built-in **tee** command is that the built-in **tee** works even if you do not have the Unix **tee** available. The built-in **tee** also logs everything that is printed on the screen, whereas the Unix **tee** used with **pager** does not log quite that much. Additionally, **tee** file logging can be turned on and off interactively from within **mysql**. This is useful when you want to log some queries to a file, but not others.

The default `mysql>` prompt can be reconfigured. The string for defining the prompt can contain the following special sequences:

Option	Description
\v	The server version
\d	The default database
\h	The server host
\p	The current TCP/IP port or socket file
\u	Your username
\U	Your full <code>user_name@host_name</code> account name
\\	A literal ‘\’ backslash character
\n	A newline character
\t	A tab character
\	A space (a space follows the backslash)
_	A space
\R	The current time, in 24-hour military time (0-23)
\r	The current time, standard 12-hour time (1-12)
\m	Minutes of the current time
\y	The current year, two digits
\Y	The current year, four digits
\D	The full current date
\s	Seconds of the current time
\w	The current day of the week in three-letter format (Mon, Tue, ...)
\P	am/pm

\o	The current month in numeric format
\O	The current month in three-letter format (Jan, Feb, ...)
\c	A counter that increments for each statement you issue
\l	The current delimiter. (New in 5.0.25)
\S	Semicolon
\'	Single quote
\"	Double quote

'\' followed by any other letter just becomes that letter.

If you specify the prompt command with no argument, **mysql** resets the prompt to the default of `mysql>`.

You can set the prompt in several ways:

- *Use an environment variable.* You can set the `MYSQL_PS1` environment variable to a prompt string. For example:

```
shell> export MYSQL_PS1="(\\u@\\h) [\\d]> "
```

- *Use a command-line option.* You can set the `--prompt` option on the command line to **mysql**. For example:

```
shell> mysql --prompt="(\\u@\\h) [\\d]> "
(user@host) [database]>
```

- *Use an option file.* You can set the prompt option in the `[mysql]` group of any MySQL option file, such as `/etc/my.cnf` or the `.my.cnf` file in your home directory. For example:

```
[mysql]
prompt=(\\u@\\h) [\\d]>\\_
```

In this example, note that the backslashes are doubled. If you set the prompt using the `prompt` option in an option file, it is advisable to double the backslashes when using the special prompt options. There is some overlap in the set of allowable prompt options and the set of special escape sequences that are recognized in option files. (These sequences are listed in [Section 4.3.2, “Using Option Files”](#).) The overlap may cause you problems

if you use single backslashes. For example, `\s` is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in `HH:MM:SS` format:

```
[mysql]
prompt="\r:\m:\s> "
```

- *Set the prompt interactively.* You can change your prompt interactively by using the `prompt` (or `\R`) command. For example:

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

8.6.3. mysql Server-Side Help

```
mysql> help search_string
```

If you provide an argument to the `help` command, **mysql** uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. The proper operation of this command requires that the help tables in the `mysql` database be initialized with help topic information (see [Section 5.2.7, “MySQL Server-Side Help Support”](#)).

If there is no match for the search string, the search fails:

```
mysql> help me
```

```
Nothing found
```

```
Please try to run 'help contents' for a list of all accessible topic
```

Use **help contents** to see a list of the help categories:

```
mysql> help contents
```

```
You asked for help about help category: "Contents"
```

```
For more information, type 'help <item>', where <item> is one of the following categories:
```

```
Account Management
Administration
Data Definition
```

- Data Manipulation
- Data Types
- Functions
- Functions and Modifiers for Use with GROUP BY
- Geographic Features
- Language Structure
- Storage Engines
- Stored Routines
- Table Maintenance
- Transactions
- Triggers

If the search string matches multiple items, **mysql** shows a list of matching topics:

```
mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
SHOW
SHOW BINARY LOGS
SHOW ENGINE
SHOW LOGS
```

Use a topic as the search string to see the help entry for that topic:

```
mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in [purge-master-logs], that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name          | File_size |
+-----+-----+
| binlog.000015    |    724935 |
| binlog.000016    |    733481 |
+-----+-----+
```

8.6.4. Executing SQL Statements from a Text File

The **mysql** client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell **mysql** to read its input from that file. To do so, create a text file *text_file* that contains the statements you wish to execute. Then invoke **mysql** as shown here:

```
shell> mysql db_name < text_file
```

If you place a `USE db_name` statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are already running **mysql**, you can execute an SQL script file using the `source` or `\.` command:

```
mysql> source file_name  
mysql> \. file_name
```

Sometimes you may want your script to display progress information to the user. For this you can insert statements like this:

```
SELECT '<info_to_display>' AS ' ';
```

The statement shown outputs `<info_to_display>`.

For more information about batch mode, see [Section 3.5, “Using **mysql** in Batch Mode”](#).

8.6.5. **mysql** Tips

This section describes some techniques that can help you use **mysql** more effectively.

8.6.5.1. Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with `\G` instead of a semicolon. For example, longer text

values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
***** 1. row *****
  msg_nro: 3068
    date: 2000-03-01 23:29:50
  time_zone: +0200
  mail_from: Monty
    reply: monty@no.spam.com
  mail_to: "Thimble Smith" <tim@no.spam.com>
    sbj: UTF-8
    txt: >>>> "Thimble" == Thimble Smith writes:
```

```
Thimble> Hi. I think this is a good idea. Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.
```

Yes, please do that.

```
Regards,
Monty
  file: inbox-jani-1
  hash: 190402944
1 row in set (0.09 sec)
```

8.6.5.2. Using the `--safe-updates` Option

For beginners, a useful startup option is `--safe-updates` (or `--i-am-a-dummy`, which has the same effect). It is helpful for cases when you might have issued a `DELETE FROM tbl_name` statement but forgotten the `WHERE` clause. Normally, such a statement deletes all rows from the table. With `--safe-updates`, you can delete rows only by specifying the key values that identify them. This helps prevent accidents.

When you use the `--safe-updates` option, **mysql** issues the following statement when it connects to the MySQL server:

```
SET SQL_SAFE_UPDATES=1,SQL_SELECT_LIMIT=1000, SQL_MAX_JOIN_SIZE=1000
```

See [Section 13.5.3, “SET Syntax”](#).

The SET statement has the following effects:

- You are not allowed to execute an UPDATE or DELETE statement unless you

specify a key constraint in the `WHERE` clause or provide a `LIMIT` clause (or both). For example:

```
UPDATE tbl_name SET not_key_column=val WHERE key_column=val;
```

```
UPDATE tbl_name SET not_key_column=val LIMIT 1;
```

- The server limits all large `SELECT` results to 1,000 rows unless the statement includes a `LIMIT` clause.
- The server aborts multiple-table `SELECT` statements that probably need to examine more than 1,000,000 row combinations.

To specify limits different from 1,000 and 1,000,000, you can override the defaults by using the `--select_limit` and `--max_join_size` options:

```
shell>mysql --safe-updates --select_limit=500 --max_join_size=10000
```

8.6.5.3. Disabling mysql Auto-Reconnect

If the **mysql** client loses its connection to the server while sending a query, it immediately and automatically tries to reconnect once to the server and send the query again. However, even if **mysql** succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables. Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+-----+
| a     |
```

```
+-----+
| NULL |
+-----+
1 row in set (0.05 sec)
```

The @a user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have **mysql** terminate with an error if the connection has been lost, you can start the **mysql** client with the --skip-reconnect option.

8.7. `mysql_explain_log` — Use EXPLAIN on Statements in Query Log

`mysql_explain_log` reads its standard input for query log contents. It uses EXPLAIN to analyze SELECT statements found in the input. UPDATE statements are rewritten to SELECT statements and also analyzed with EXPLAIN.

`mysql_explain_log` then displays a summary of its results.

The results may assist you in determining which queries result in table scans and where it would be beneficial to add indexes to your tables.

Invoke `mysql_explain_log` like this, where *log_file* contains all or part of a MySQL query log:

```
shell> mysql_explain_log [options] < log_file
```

`mysql_explain_log` understands the following options:

- `--date=YYMMDD, -d YYMMDD`

Select entries from the log only for the given date.

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--password=password, -ppassword`

The password to use when connecting to the server.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- `--printerror=1, -e 1`

Enable error output.

- `--socket=path, -S path`

For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

8.8. `mysqlaccess` — Client for Checking Access Privileges

`mysqlaccess` is a diagnostic tool that Yves Carlier has provided for the MySQL distribution. It checks the access privileges for a hostname, username, and database combination. Note that `mysqlaccess` checks access using only the user, db, and host tables. It does not check table, column, or routine privileges specified in the `tables_priv`, `columns_priv`, or `procs_priv` tables.

Invoke `mysqlaccess` like this:

```
shell> mysqlaccess [host_name [user_name [db_name]]] [options]
```

`mysqlaccess` understands the following options:

- `--help, -?`

Display a help message and exit.

- `--brief, -b`

Generate reports in single-line tabular format.

- `--commit`

Copy the new access privileges from the temporary tables to the original grant tables. The grant tables must be flushed for the new privileges to take effect. (For example, execute a `mysqladmin reload` command.)

- `--copy`

Reload the temporary grant tables from original ones.

- `--db=db_name, -d db_name`

Specify the database name.

- `--debug=N`

Specify the debug level. *n* can be an integer from 0 to 3.

- --host=host_name, -h host_name

The hostname to use in the access privileges.

- --howto

Display some examples that show how to use **mysqlaccess**.

- --old_server

Assume that the server is an old MySQL server (before MySQL 3.21) that does not yet know how to handle full WHERE clauses.

- --password[=password], -p[password]

The password to use when connecting to the server. If you omit the *password* value following the --password or -p option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- --plan

Display suggestions and ideas for future releases.

- --preview

Show the privilege differences after making changes to the temporary grant tables.

- --relnotes

Display the release notes.

- --rhost=host_name, -H host_name

Connect to the MySQL server on the given host.

- `--rollback`

Undo the most recent changes to the temporary grant tables.

- `--spassword[=password], -P[password]`

The password to use when connecting to the server as the superuser. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- `--superuser=user_name, -U user_name`

Specify the username for connecting as the superuser.

- `--table, -t`

Generate reports in table format.

- `--user=user_name, -u user_name`

The username to use in the access privileges.

- `--version, -v`

Display version information and exit.

If your MySQL distribution is installed in some non-standard location, you must change the location where **mysqlaccess** expects to find the **mysql** client. Edit the `mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL      = '/usr/local/bin/mysql';    # path to mysql executable
```

Change the path to reflect the location where **mysql** actually is stored on your system. If you do not do this, a Broken pipe error will occur when you run **mysqlaccess**.

8.9. `mysqladmin` — Client for Administering a MySQL Server

mysqladmin is a client for performing administrative operations. You can use it to check the server's configuration and current status, to create and drop databases, and more.

Invoke **mysqladmin** like this:

```
shell> mysqladmin [options] command [command-arg] [command [command-
```

mysqladmin supports the commands described in the following list. Some of the commands take an argument following the command name.

- `create db_name`

Create a new database named *db_name*.

- `debug`

Tell the server to write debug information to the error log.

- `drop db_name`

Delete the database named *db_name* and all its tables.

- `extended-status`

Display the server status variables and their values.

- `flush-hosts`

Flush all information in the host cache.

- `flush-logs`

Flush all logs.

- `flush-privileges`

Reload the grant tables (same as `reload`).

- `flush-status`

Clear status variables.

- `flush-tables`

Flush all tables.

- `flush-threads`

Flush the thread cache.

- `kill id,id,...`

Kill server threads. If multiple thread ID values are given, there must be no spaces in the list.

- `old-password new-password`

This is like the `password` command but stores the password using the old (pre-4.1) password-hashing format. (See [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#).)

- `password new-password`

Set a new password. This changes the password to *new-password* for the account that you use with **mysqladmin** for connecting to the server. Thus, the next time you invoke **mysqladmin** (or any other client program) using the same account, you will need to specify the new password.

If the *new-password* value contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotes. On Windows, be sure to use double quotes rather than single quotes; single quotes are not stripped from the password, but rather are interpreted as part of the password. For example:

```
shell> mysqladmin password "my new password"
```

- `ping`

Check whether the server is alive. The return status from **mysqladmin** is 0 if the server is running, 1 if it is not. This is 0 even in case of an error such as Access denied, because this means that the server is running but refused the connection, which is different from the server not running.

- processlist

Show a list of active server threads. This is like the output of the SHOW PROCESSLIST statement. If the --verbose option is given, the output is like that of SHOW FULL PROCESSLIST. (See [Section 13.5.4.19, “SHOW PROCESSLIST Syntax”](#).)

- reload

Reload the grant tables.

- refresh

Flush all tables and close and open log files.

- shutdown

Stop the server.

- start-slave

Start replication on a slave server.

- status

Display a short server status message.

- stop-slave

Stop replication on a slave server.

- variables

Display the server system variables and their values.

- version

Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | User  | Host          | db | Command | Time | State | Info
+-----+-----+-----+-----+-----+-----+-----+-----+
| 51 | monty | localhost    |    | Query   | 0    |      | show proces
+-----+-----+-----+-----+-----+-----+-----+-----+
Uptime: 1473624  Threads: 1  Questions: 39487
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

The **mysqladmin status** command result displays the following values:

- Uptime

The number of seconds the MySQL server has been running.

- Threads

The number of active threads (clients).

- Questions

The number of questions (queries) from clients since the server was started.

- Slow queries

The number of queries that have taken more than `long_query_time` seconds. See [Section 5.12.4, “The Slow Query Log”](#).

- Opens

The number of tables the server has opened.

- Flush tables

The number of `flush-*`, `refresh`, and `reload` commands the server has executed.

- Open tables

The number of tables that currently are open.

- Memory in use

The amount of memory allocated directly by **mysqld**. This value is displayed only when MySQL has been compiled with `--with-debug=full`.

- Maximum memory used

The maximum amount of memory allocated directly by **mysqld**. This value is displayed only when MySQL has been compiled with `--with-debug=full`.

If you execute **mysqladmin shutdown** when connecting to a local server using a Unix socket file, **mysqladmin** waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

mysqladmin supports the following options:

- `--help, -?`

Display a help message and exit.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--count=N, -c N`

The number of iterations to make for repeated command execution. This works only with the `--sleep` option.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The *debug_options* string often is

'd:t:o,file_name'. The default is 'd:t:o,/tmp/mysqladmin.trace'.

- --default-character-set=charset_name

Use *charset_name* as the default character set. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- --force, -f

Do not ask for confirmation for the drop db_name command. With multiple commands, continue even if an error occurs.

- --host=host_name, -h host_name

Connect to the MySQL server on the given host.

- --password[=password], -p[password]

The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- --port=port_num, -P port_num

The TCP/IP port number to use for the connection.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

The connection protocol to use.

- --relative, -r

Show the difference between the current and previous values when used with the --sleep option. Currently, this option works only with the extended-status command.

- --silent, -s

Exit silently if a connection to the server cannot be established.

- `--sleep=delay, -i delay`

Execute commands repeatedly, sleeping for *delay* seconds in between. The `--count` option determines the number of iterations.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.9.7.3, “SSL Command Options”](#).

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does.

- `--version, -V`

Display version information and exit.

- `--vertical, -E`

Print output vertically. This is similar to `--relative`, but prints output vertically.

- `--wait[=count], -w[count]`

If the connection cannot be established, wait and retry instead of aborting. If a *count* value is given, it indicates the number of times to retry. The default is one time.

You can also set the following variables by using `--var_name=value` syntax:

- `connect_timeout`

The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- `shutdown_timeout`

The maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

8.10. **mysqlbinlog** — Utility for Processing Binary Log Files

The binary log files that the server generates are written in binary format. To examine these files in text format, use the **mysqlbinlog** utility. You can also use **mysqlbinlog** to read relay log files written by a slave server in a replication setup. Relay logs have the same format as binary log files.

Invoke **mysqlbinlog** like this:

```
shell> mysqlbinlog [options] log_file ...
```

For example, to display the contents of the binary log file named `binlog.000003`, use this command:

```
shell> mysqlbinlog binlog.000003
```

The output includes all events contained in `binlog.000003`. Event information includes the statement executed, the time the statement took, the thread ID of the client that issued it, the timestamp when it was executed, and so forth.

The output from **mysqlbinlog** can be re-executed (for example, by using it as input to **mysql**) to reapply the statements in the log. This is useful for recovery operations after a server crash. For other usage examples, see the discussion later in this section.

Normally, you use **mysqlbinlog** to read binary log files directly and apply them to the local MySQL server. It is also possible to read binary logs from a remote server by using the `--read-from-remote-server` option. When you read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`; they are ignored except when you also use the `--read-from-remote-server` option.

Binary logs and relay logs are discussed further in [Section 5.12.3, “The Binary Log”](#), and [Section 6.3.4, “Replication Relay and Status Files”](#).

mysqlbinlog supports the following options:

- `--help, -?`

Display a help message and exit.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--database=db_name, -d db_name`

List entries for just this database (local log only). You can only specify one database with this option - if you specify multiple `--database` options, only the last one is used. This option forces **mysqlbinlog** to output entries from the binary log where the default database (that is, the one selected by USE) is *db_name*. Note that this does not replicate cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` while having selected a different database or no database.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. A typical *debug_options* string is often `'d:t:o,file_name'`.

- `--disable-log-bin, -D`

Disable binary logging. This is useful for avoiding an endless loop if you use the `--to-last-log` option and are sending the output to the same MySQL server. This option also is useful when restoring after a crash to avoid duplication of the statements you have logged.

This option requires that you have the SUPER privilege. It causes **mysqlbinlog** to include a `SET SQL_LOG_BIN=0` statement in its output to disable binary logging of the remaining output. The SET statement is ineffective unless you have the SUPER privilege.

- `--force-read, -f`

With this option, if **mysqlbinlog** reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this

option, **mysqlbinlog** stops if it reads such an event.

- --hexdump, -H

Display a hex dump of the log in comments. This output can be helpful for replication debugging. Hex dump format is discussed later in this section. This option was added in MySQL 5.0.16.

- --host=host_name, -h host_name

Get the binary log from the MySQL server on the given host.

- --local-load=path, -l path

Prepare local temporary files for LOAD DATA INFILE in the specified directory.

- --offset=N, -o N

Skip the first N entries in the log.

- --password[=password], -p[password]

The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- --port=port_num, -P port_num

The TCP/IP port number to use for connecting to a remote server.

- --position=N, -j N

Deprecated. Use --start-position instead.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

The connection protocol to use.

- `--read-from-remote-server, -R`

Read the binary log from a MySQL server rather than reading a local log file. Any connection parameter options are ignored unless this option is given as well. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`.

- `--result-file=name, -r name`

Direct output to the given file.

- `--set-charset=charset_name`

Add a `SET NAMES charset_name` statement to the output to specify the character set to be used for processing log files. This option was added in MySQL 5.0.23.

- `--short-form, -s`

Display only the statements contained in the log, without any extra information.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--start-datetime=datetime`

Start reading the binary log at the first event having a timestamp equal to or later than the *datetime* argument. The *datetime* value is relative to the local time zone on the machine where you run **mysqlbinlog**. The value should be in a format accepted for the DATETIME or TIMESTAMP data types. For example:

```
shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog
```

This option is useful for point-in-time recovery. See [Section 5.10.2, “Example Backup and Recovery Strategy”](#).

- `--stop-datetime=datetime`

Stop reading the binary log at the first event having a timestamp equal or posterior to the *datetime* argument. This option is useful for point-in-time recovery. See the description of the `--start-datetime` option for information about the *datetime* value.

- `--start-position=N`

Start reading the binary log at the first event having a position equal to the *N* argument. This option applies to the first log file named on the command line.

- `--stop-position=N`

Stop reading the binary log at the first event having a position equal or greater than the *N* argument. This option applies to the last log file named on the command line.

- `--to-last-log, -t`

Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server`.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to a remote server.

- `--version, -V`

Display version information and exit.

You can also set the following variable by using `--var_name=value` syntax:

- `open_files_limit`

Specify the number of open file descriptors to reserve.

It is also possible to set variables by using `--set-variable=var_name=value` or

-O var_name=value syntax. *This syntax is deprecated.*

You can pipe the output of **mysqlbinlog** into the **mysql** client to execute the statements contained in the binary log. This is used to recover from a crash when you have an old backup (see [Section 5.10.1, “Database Backups”](#)). For example:

```
shell> mysqlbinlog binlog.000001 | mysql
```

Or:

```
shell> mysqlbinlog binlog.[0-9]* | mysql
```

You can also redirect the output of **mysqlbinlog** to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the **mysql** program.

mysqlbinlog has the `--start-position` option, which prints only those statements with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the `--stop-datetime` option (to be able to say, for example, “roll forward my databases to how they were today at 10:30 a.m.”).

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql # DANGER!!  
shell> mysqlbinlog binlog.000002 | mysql # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first **mysql** process terminates, the server drops the temporary table. When the second **mysql** process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 > /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -e "source /tmp/statements.sql"
```

mysqlbinlog can produce output that reproduces a `LOAD DATA INFILE` operation without the original data file. **mysqlbinlog** copies the data to a temporary file and writes a `LOAD DATA LOCAL INFILE` statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the `--local-load` option.

Because **mysqlbinlog** converts `LOAD DATA INFILE` statements to `LOAD DATA LOCAL INFILE` statements (that is, it adds `LOCAL`), both the client and the server that you use to process the statements must be configured to allow `LOCAL` capability. See [Section 5.7.4, “Security Issues with `LOAD DATA LOCAL`”](#).

Warning: The temporary files created for `LOAD DATA LOCAL` statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like *original_file_name-#-#*.

The `--hexdump` option produces a hex dump of the log contents in comments:

```
shell> mysqlbinlog --hexdump master-bin.000001
```

With the preceding command, the output might look like this:

```
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=
# at 4
#051024 17:24:13 server id 1  end_log_pos 98
# Position  Timestamp  Type  Master ID      Size      Master Pos
# 00000004  9d fc 5c 43   0f   01 00 00 00    5e 00 00 00    62 00 00 0
# 00000017  04 00 35 2e 30 2e 31 35  2d 64 65 62 75 67 2d 6c |..5.0.1
# 00000027  6f 67 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |og.....
# 00000037  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |.....
# 00000047  00 00 00 00 9d fc 5c 43  13 38 0d 00 08 00 12 00 |.....
# 00000057  04 04 04 04 12 00 00 4b  00 04 1a      |.....
#          Start: binlog v 4, server v 5.0.15-debug-log created 051024
#          at startup
```

ROLLBACK;

Hex dump output currently contains the following elements. This format might change in the future.

- **Position:** The byte position within the log file.
- **Timestamp:** The event timestamp. In the example shown, '9d fc 5c 43' is the representation of '051024 17:24:13' in hexadecimal.
- **Type:** The type of the log event. In the example shown, '0f' means that the example event is a `FORMAT_DESCRIPTION_EVENT`. The following table lists the possible types.

Type	Name	Meaning
00	UNKNOWN_EVENT	This event should never be present in the log.
01	START_EVENT_V3	This indicates the start of a log file written by MySQL 4 or earlier.
02	QUERY_EVENT	The most common type of events. These contain statements executed on the master.
03	STOP_EVENT	Indicates that master has stopped.
04	ROTATE_EVENT	Written when the master switches to a new log file.
05	INTVAR_EVENT	Used mainly for <code>AUTO_INCREMENT</code> values and when the <code>LAST_INSERT_ID()</code> function is used in the statement.
06	LOAD_EVENT	Used for <code>LOAD DATA INFILE</code> in MySQL 3.23.
07	SLAVE_EVENT	Reserved for future use.
08	CREATE_FILE_EVENT	Used for <code>LOAD DATA INFILE</code> statements. This indicates the start of execution of such a statement. A temporary file is created on the slave. Used in MySQL 4 only.

09	APPEND_BLOCK_EVENT	Contains data for use in a LOAD DATA INFILE statement. The data is stored in the temporary file on the slave.
0a	EXEC_LOAD_EVENT	Used for LOAD DATA INFILE statements. The contents of the temporary file is stored in the table on the slave. Used in MySQL 4 only.
0b	DELETE_FILE_EVENT	Rollback of a LOAD DATA INFILE statement. The temporary file should be deleted on slave.
0c	NEW_LOAD_EVENT	Used for LOAD DATA INFILE in MySQL 4 and earlier.
0d	RAND_EVENT	Used to send information about random values if the RAND() function is used in the statement.
0e	USER_VAR_EVENT	Used to replicate user variables.
0f	FORMAT_DESCRIPTION_EVENT	This indicates the start of a log file written by MySQL 5 or later.
10	XID_EVENT	Event indicating commit of an XA transaction.
11	BEGIN_LOAD_QUERY_EVENT	Used for LOAD DATA INFILE statements in MySQL 5 and later.
12	EXECUTE_LOAD_QUERY_EVENT	Used for LOAD DATA INFILE statements in MySQL 5 and later.
13	TABLE_MAP_EVENT	Reserved for future use.
14	WRITE_ROWS_EVENT	Reserved for future use.
15	UPDATE_ROWS_EVENT	Reserved for future use.
16	DELETE_ROWS_EVENT	Reserved for future use.

- Master ID: The server id of the master that created the event.
- Size: The size in bytes of the event.
- Master Pos: The position of the event in the original master log file.
- Flags: 16 flags. Currently, the following flags are used. The others are

reserved for the future.

Flag	Name	Meaning
01	LOG_EVENT_BINLOG_IN_USE_F	Log file correctly closed. (Used only in FORMAT_DESCRIPTION_EVENT.) If this flag is set (if the flags are, for example, '01 00') in a FORMAT_DESCRIPTION_EVENT, the log file has not been properly closed. Most probably this is because of a master crash (for example, due to power failure).
02		Reserved for future use.
04	LOG_EVENT_THREAD_SPECIFIC_F	Set if the event is dependent on the connection it was executed in (for example, '04 00'), for example, if the event uses temporary tables.
08	LOG_EVENT_SUPPRESS_USE_F	Set in some circumstances when the event is not dependent on the default database.

The other flags are reserved for future use.

8.11. `mysqlcheck` — A Table Maintenance and Repair Program

The `mysqlcheck` client checks, repairs, optimizes, and analyzes tables.

`mysqlcheck` is similar in function to `myisamchk`, but works differently. The main operational difference is that `mysqlcheck` must be used when the `mysqld` server is running, whereas `myisamchk` should be used when it is not. The benefit of using `mysqlcheck` is that you do not have to stop the server to check or repair your tables.

`mysqlcheck` uses the SQL statements `CHECK TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, and `OPTIMIZE TABLE` in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statements to the server to be executed. For details about which storage engines each statement works with, see the descriptions for those statements in [Chapter 13, SQL Statement Syntax](#).

The MyISAM storage engine supports all four statements, so `mysqlcheck` can be used to perform all four operations on MyISAM tables. Other storage engines do not necessarily support all operations. In such cases, an error message is displayed. For example, if `test.t` is a MEMORY table, an attempt to check it produces this result:

```
shell> mysqlcheck test t
test.t
note      : The storage engine for the table doesn't support check
```

There are three general ways to invoke `mysqlcheck`:

```
shell> mysqlcheck [options] db_name [tables]
shell> mysqlcheck [options] --databases db_name1 [db_name2 db_name3.]
shell> mysqlcheck [options] --all-databases
```

If you do not name any tables following `db_name` or if you use the `--databases` or `--all-databases` option, entire databases are checked.

`mysqlcheck` has a special feature compared to other client programs. The default behavior of checking tables (`--check`) can be changed by renaming the binary. If

you want to have a tool that repairs tables by default, you should just make a copy of **mysqlcheck** named **mysqlrepair**, or make a symbolic link to **mysqlcheck** named **mysqlrepair**. If you invoke **mysqlrepair**, it repairs tables.

The following names can be used to change **mysqlcheck** default behavior:

mysqlrepair	The default option is <code>--repair</code>
mysqlanalyze	The default option is <code>--analyze</code>
mysqloptimize	The default option is <code>--optimize</code>

mysqlcheck supports the following options:

- `--help, -?`

Display a help message and exit.

- `--all-databases, -A`

Check all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--all-in-1, -1`

Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze, -a`

Analyze the tables.

- `--auto-repair`

If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- --check, -c

Check the tables for errors. This is the default operation.

- --check-only-changed, -C

Check only tables that have changed since the last check or that have not been closed properly.

- --check-upgrade, -g

Invoke CHECK TABLE with the FOR UPGRADE option to check tables for incompatibilities with the current version of the server. This option was added in MySQL 5.0.19.

- --compress

Compress all information sent between the client and the server if both support compression.

- --databases, -B

Process all tables in the named databases. Normally, **mysqlcheck** treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names.

- --debug[=debug_options], -# [debug_options]

Write a debugging log. A typical *debug_options* string is often 'd:t:o,file_name'.

- --default-character-set=charset_name

Use *charset_name* as the default character set. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- --extended, -e

If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- `--fast, -F`

Check only tables that have not been closed properly.

- `--force, -f`

Continue even if an SQL error occurs.

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--medium-check, -m`

Do a check that is faster than an `--extended` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--optimize, -o`

Optimize the tables.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--quick, -q`

If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- `--repair, -r`

Perform a repair that can fix almost anything except unique keys that are not unique.

- `--silent, -s`

Silent mode. Print only error messages.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.9.7.3, “SSL Command Options”](#).

- `--tables`

Overrides the `--databases` or `-B` option. All name arguments following the option are regarded as table names.

- `--use-frm`

For repair operations on MyISAM tables, get the table structure from the `.frm` file so that the table can be repaired even if the `.MYI` header is corrupted.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print information about the various stages of program operation.

- `--version, -V`

Display version information and exit.

8.12. mysqldump — A Database Backup Program

The **mysqldump** client is a backup program originally written by Igor Romanenko. It can be used to dump a database or a collection of databases for backup or for transferring the data to another SQL server (not necessarily a MySQL server). The dump contains SQL statements to create the table or populate it, or both.

If you are doing a backup on the server, and your tables all are MyISAM tables, consider using the **mysqlhotcopy** instead because it can accomplish faster backups and faster restores. See [Section 8.13, “mysqlhotcopy — A Database Backup Program”](#).

There are three general ways to invoke **mysqldump**:

```
shell> mysqldump [options] db_name [tables]
shell> mysqldump [options] --databases db_name1 [db_name2 db_name3..
shell> mysqldump [options] --all-databases
```

If you do not name any tables following *db_name* or if you use the `--databases` or `--all-databases` option, entire databases are dumped.

To get a list of the options your version of **mysqldump** supports, execute **mysqldump --help**.

If you run **mysqldump** without the `--quick` or `--opt` option, **mysqldump** loads the whole result set into memory before dumping the result. This can be a problem if you are dumping a big database. The `--opt` option is enabled by default, but can be disabled with `--skip-opt`.

If you are using a recent copy of the **mysqldump** program to generate a dump to be reloaded into a very old MySQL server, you should not use the `--opt` or `--extended-insert` option. Use `--skip-opt` instead.

mysqldump supports the following options:

- `--help, -?`

Display a help message and exit.

- --add-drop-database

Add a DROP DATABASE statement before each CREATE DATABASE statement.

- --add-drop-table

Add a DROP TABLE statement before each CREATE TABLE statement.

- --add-locks

Surround each table dump with LOCK TABLES and UNLOCK TABLES statements. This results in faster inserts when the dump file is reloaded. See [Section 7.2.16, “Speed of INSERT Statements”](#).

- --all-databases, -A

Dump all tables in all databases. This is the same as using the --databases option and naming all the databases on the command line.

- --allow-keywords

Allow creation of column names that are keywords. This works by prefixing each column name with the table name.

- --character-sets-dir=path

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- --comments, -i

Write additional information in the dump file such as program version, server version, and host. . This option is enabled by default. To suppress additional, use --skip-comments.

- --compact

Produce less verbose output. This option suppresses comments and enables the --skip-add-drop-table, --no-set-names, --skip-disable-keys, and --skip-add-locks options.

- `--compatible=name`

Produce output that is more compatible with other database systems or with older MySQL servers. The value of `name` can be `ansi`, `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`, `db2`, `maxdb`, `no_key_options`, `no_table_options`, or `no_field_options`. To use several values, separate them by commas. These values have the same meaning as the corresponding options for setting the server SQL mode. See [Section 5.2.5, “The Server SQL Mode”](#).

This option does not guarantee compatibility with other servers. It only enables those SQL mode values that are currently available for making dump output more compatible. For example, `--compatible=oracle` does not map data types to Oracle types or use Oracle comment syntax.

- `--complete-insert, -c`

Use complete `INSERT` statements that include column names.

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--create-options`

Include all MySQL-specific table options in the `CREATE TABLE` statements.

- `--databases, -B`

Dump several databases. Normally, **mysqldump** treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE` and `USE` statements are included in the output before each new database.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The `debug_options` string is often `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysqldump.trace'`.

- `--default-character-set=charset_name`

Use *charset_name* as the default character set. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#). If not specified, **mysqldump** uses utf8.

- `--delayed-insert`

Write `INSERT DELAYED` statements rather than `INSERT` statements.

- `--delete-master-logs`

On a master replication server, delete the binary logs after performing the dump operation. This option automatically enables `--master-data`.

- `--disable-keys, -K`

For each table, surround the `INSERT` statements with `/*!40000 ALTER TABLE tbl_name DISABLE KEYS */;` and `/*!40000 ALTER TABLE tbl_name ENABLE KEYS */;` statements. This makes loading the dump file faster because the indexes are created after all rows are inserted. This option is effective for MyISAM tables only.

- `--extended-insert, -e`

Use multiple-row `INSERT` syntax that include several `VALUES` lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`, `--lines-terminated-by=...`

These options are used with the `-T` option and have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).

- `--first-slave, -x`

Deprecated. Now renamed to `--lock-all-tables`.

- `--flush-logs, -F`

Flush the MySQL server log files before starting the dump. This option requires the RELOAD privilege. Note that if you use this option in combination with the `--all-databases` (or `-A`) option, the logs are flushed *for each database dumped*. The exception is when using `--lock-all-tables` or `--master-data`: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked. If you want your dump and the log flush to happen at exactly the same moment, you should use `--flush-logs` together with either `--lock-all-tables` or `--master-data`.

- `--force, -f`

Continue even if an SQL error occurs during a table dump.

One use for this option is to cause **mysqldump** to continue executing even when it encounters a view that has become invalid because the definition refers to a table that has been dropped. Without `--force`, **mysqldump** exits with an error message. With `--force`, **mysqldump** prints the error message, but it also writes a SQL comment containing the view definition to the dump output and continues executing.

- `--host=host_name, -h host_name`

Dump data from the MySQL server on the given host. The default host is `localhost`.

- `--hex-blob`

Dump binary columns using hexadecimal notation (for example, `'abc'` becomes `0x616263`). The affected data types are `BINARY`, `VARBINARY`, and `BLOB`. As of MySQL 5.0.13, `BIT` columns are affected as well.

- `--ignore-table=db_name.tbl_name`

Do not dump the given table, which must be specified using both the database and table names. To ignore multiple tables, use this option multiple times.

- `--insert-ignore`

Write `INSERT` statements with the `IGNORE` option.

- `--lock-all-tables, -x`

Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off `--single-transaction` and `--lock-tables`.

- `--lock-tables, -l`

Lock all tables before starting the dump. The tables are locked with `READ LOCAL` to allow concurrent inserts in the case of `MyISAM` tables. For transactional tables such as `InnoDB` and `BDB`, `--single-transaction` is a much better option, because it does not need to lock the tables at all.

Please note that when dumping multiple databases, `--lock-tables` locks tables for each database separately. So, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

- `--master-data[=value]`

Write the binary log filename and position to the output. This option requires the `RELOAD` privilege and the binary log must be enabled. If the option value is equal to 1, the position and filename are written to the dump output in the form of a `CHANGE MASTER` statement that makes a slave server start from the correct position in the master's binary logs if you use this SQL dump of the master to set up a slave. If the option value is equal to 2, the `CHANGE MASTER` statement is written as an SQL comment. This is the default action if *value* is omitted.

The `--master-data` option turns on `--lock-all-tables`, unless `--single-transaction` also is specified (in which case, a global read lock is only acquired a short time at the beginning of the dump. See also the description for `--single-transaction`). In all cases, any action on logs happens at the exact moment of the dump. This option automatically turns off `--lock-tables`.

- `--no-autocommit`

Enclose the `INSERT` statements for each dumped table within `SET AUTOCOMMIT=0` and `COMMIT` statements.

- `--no-create-db, -n`

This option suppresses the `CREATE DATABASE` statements that are otherwise included in the output if the `--databases` or `--all-databases` option is given.

- `--no-create-info, -t`

Do not write `CREATE TABLE` statements that re-create each dumped table.

- `--no-data, -d`

Do not write any row information for the table. This is very useful if you want to dump only the `CREATE TABLE` statement for the table.

- `--opt`

This option is shorthand; it is the same as specifying `--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`. It should give you a fast dump operation and produce a dump file that can be reloaded into a MySQL server quickly.

The `--opt` option is enabled by default. To disable the options that it enables, use `--skip-opt`. To disable only certain of the options enabled by `--opt`, use their `--skip` forms; for example, `--skip-add-drop-table` or `--skip-quick`. Alternatively, use `--skip-opt` to disable the options enabled by `--opt`, followed by options to enable the features that you want. Options are processed in order, so the options to enable features must follow `--skip-opt`. For example, `--skip-opt --extended-insert` enables extended inserts, but `--extended-insert --skip-opt` does not.

- `--order-by-primary`

Sorts each table's rows by its primary key, or its first unique index, if such

an index exists. This is useful when dumping a MyISAM table to be loaded into an InnoDB table, but will make the dump itself take considerably longer.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--quick, -q`

This option is useful for dumping large tables. It forces **mysqldump** to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- `--quote-names, -Q`

Quote database, table, and column names within ‘`’ characters. If the `ANSI_QUOTES` SQL mode is enabled, names are quoted within ‘”’ characters. This option is enabled by default. It can be disabled with `--skip-quote-names`, but this option should be given after any option such as `--compatible` that may enable `--quote-names`.

- `--result-file=file, -r file`

Direct output to a given file. This option should be used on Windows to

prevent newline ‘\n’ characters from being converted to ‘\r\n’ carriage return/newline sequences. The result file is created and its contents overwritten, even if an error occurs while generating the dump. The previous contents are lost.

- `--routines, -R`

Dump stored routines (functions and procedures) from the dumped databases. The output generated by using `--routines` contains `CREATE PROCEDURE` and `CREATE FUNCTION` statements to re-create the routines. However, these statements do not include attributes such as the routine creation and modification timestamps. This means that when the routines are reloaded, they will be created with the timestamps equal to the reload time.

If you require routines to be re-created with their original timestamp attributes, do not use `--routines`. Instead, dump and reload the contents of the `mysql.proc` table directly, using a MySQL account that has appropriate privileges for the `mysql` database.

This option was added in MySQL 5.0.13. Before that, stored routines are not dumped. Routine `DEFINER` values are not dumped until MySQL 5.0.20. This means that before 5.0.20, when routines are reloaded, they will be created with the definer set to the reloading user. If you require routines to be re-created with their original definer, dump and load the contents of the `mysql.proc` table directly as described earlier.

- `--set-charset`

Add `SET NAMES default_character_set` to the output. This option is enabled by default. To suppress the `SET NAMES` statement, use `--skip-set-charset`.

- `--single-transaction`

This option issues a `BEGIN SQL` statement before dumping data from the server. It is useful only with transactional tables such as `InnoDB` and `BDB`, because then it dumps the consistent state of the database at the time when `BEGIN` was issued without blocking any applications.

When using this option, you should keep in mind that only InnoDB tables are dumped in a consistent state. For example, any MyISAM or MEMORY tables dumped while using this option may still change state.

The `--single-transaction` option and the `--lock-tables` option are mutually exclusive, because `LOCK TABLES` causes any pending transactions to be committed implicitly.

This option is not supported for MySQL Cluster tables; the results cannot be guaranteed to be consistent due to the fact that the `NDBCluster` storage engine supports only the `READ_COMMITTED` transaction isolation level. You should always use `NDB backup` and `restore` instead.

To dump big tables, you should combine this option with `--quick`.

- `--skip-opt`

See the description for the `--opt` option.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--skip-comments`

See the description for the `--comments` option.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.9.7.3, “SSL Command Options”](#).

- `--tab=path, -T path`

Produce tab-separated data files. For each dumped table, **mysqldump** creates a `tbl_name.sql` file that contains the `CREATE TABLE` statement that creates the table, and a `tbl_name.txt` file that contains its data. The option value is the directory in which to write the files.

By default, the .txt data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the `--fields-xxx` and `--lines--xxx` options.

Note: This option should be used only when **mysqldump** is run on the same machine as the **mysqld** server. You must have the FILE privilege, and the server must have permission to write files in the directory that you specify.

- `--tables`

Override the `--databases` or `-B` option. All name arguments following the option are regarded as table names.

- `--triggers`

Dump triggers for each dumped table. This option is enabled by default; disable it with `--skip-triggers`. This option was added in MySQL 5.0.11. Before that, triggers are not dumped.

- `--tz-utc`

Add `SET TIME_ZONE='+00:00'` to the dump file so that `TIMESTAMP` columns can be dumped and reloaded between servers in different time zones. Without this option, `TIMESTAMP` columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change. `--tz-utc` also protects against changes due to daylight saving time. `--tz-utc` is enabled by default. To disable it, use `--skip-tz-utc`. This option was added in MySQL 5.0.15.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does.

- `--version, -V`

Display version information and exit.

- `--where='where_condition', -w 'where_condition'`

Dump only rows selected by the given `WHERE` condition. Note that quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter.

Examples:

```
--where="user='jimf' "  
-w"userid>1"  
-w"userid<1"
```

- `--xml, -X`

Write dump output as well-formed XML.

You can also set the following variables by using `--var_name=value` syntax:

- `max_allowed_packet`

The maximum size of the buffer for client/server communication. The maximum is 1GB.

- `net_buffer_length`

The initial size of the buffer for client/server communication. When creating multiple-row-insert statements (as with option `--extended-insert` or `--opt`), **mysqldump** creates rows up to `net_buffer_length` length. If you increase this variable, you should also ensure that the `net_buffer_length` variable in the MySQL server is at least this large.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. *This syntax is deprecated.*

The most common use of **mysqldump** is probably for making a backup of an entire database:

```
shell> mysqldump --opt db_name > backup-file.sql
```

You can read the dump file back into the server like this:

```
shell> mysql db_name < backup-file.sql
```

Or like this:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

mysqldump is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

It is possible to dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.
```

To dump all databases, use the `--all-databases` option:

```
shell> mysqldump --all-databases > all_databases.sql
```

For InnoDB tables, `mysqldump` provides a way of making an online backup:

```
shell> mysqldump --all-databases --single-transaction > all_database
```

This backup just needs to acquire a global read lock on all tables (using `FLUSH TABLES WITH READ LOCK`) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If and only if one long updating statement is running when the `FLUSH` statement is issued, the MySQL server may get stalled until that long statement finishes, and then the dump becomes lock-free. If the update statements that the MySQL server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as “roll-forward,” when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log (see [Section 5.12.3, “The Binary Log”](#)) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mysqldump --all-databases --flush-logs --master-data=2  
> all_databases.sql
```

The simultaneous use of `--master-data` and `--single-transaction` provides a convenient way to make an online backup suitable for point-in-time recovery if tables are stored in the InnoDB storage engine.

For more information on making backups, see [Section 5.10.1, “Database Backups”](#), and [Section 5.10.2, “Example Backup and Recovery Strategy”](#).

8.13. **mysqlhotcopy** — A Database Backup Program

mysqlhotcopy is a Perl script that was originally written and contributed by Tim Bunce. It uses `LOCK TABLES`, `FLUSH TABLES`, and `cp` or `scp` to make a database backup quickly. It is the fastest way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. **mysqlhotcopy** works only for backing up MyISAM and ARCHIVE tables. It runs on Unix and NetWare.

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mysqlhotcopy db_name./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde ('~'):

```
shell> mysqlhotcopy db_name./~regex/
```

mysqlhotcopy supports the following options:

- `--help, -?`

Display a help message and exit.

- `--addtodest`

Do not rename target directory (if it exists); merely add files to it.

- `--allowold`

Do not abort if a target exists; rename it by adding an `_old` suffix.

- `--checkpoint=db_name.tbl_name`

Insert checkpoint entries into the specified database `db_name` and table `tbl_name`.

- --chroot=path

Base directory of the **chroot** jail in which **mysqld** operates. The *path* value should match that of the --chroot option given to **mysqld**.

- --debug

Enable debug output.

- --dryrun, -n

Report actions without performing them.

- --flushlog

Flush logs after all tables are locked.

- --host=host_name, -h host_name

The hostname of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to localhost using a Unix socket file.

- --keepold

Do not delete previous (renamed) target when done.

- --method=command

The method for copying files (cp or scp).

- --noindices

Do not include full index files in the backup. This makes the backup smaller and faster. The indexes for reloaded tables can be reconstructed later with **myisamchk -rq**.

- --password=password, -ppassword

The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs. You

can use an option file to avoid giving the password on the command line.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use when connecting to the local server.

- `--quiet, -q`

Be silent except for errors.

- `--record_log_pos=db_name.tbl_name`

Record master and slave status in the specified database *db_name* and table *tbl_name*.

- `--regex=expr`

Copy all databases with names that match the given regular expression.

- `--resetmaster`

Reset the binary log after locking all the tables.

- `--resetslave`

Reset the master `.info` file after locking all the tables.

- `--socket=path, -S path`

The Unix socket file to use for the connection.

- `--suffix=str`

The suffix for names of copied databases.

- `--tmpdir=path`

The temporary directory. The default is `/tmp`.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

mysqlhotcopy reads the `[client]` and `[mysqlhotcopy]` option groups from option files.

To execute **mysqlhotcopy**, you must have access to the files for the tables that you are backing up, the `SELECT` privilege for those tables, the `RELOAD` privilege (to be able to execute `FLUSH TABLES`), and the `LOCK TABLES` privilege (to be able to lock the tables).

Use `perldoc` for additional **mysqlhotcopy** documentation, including information about the structure of the tables needed for the `--checkpoint` and `--record_log_pos` options:

```
shell> perldoc mysqlhotcopy
```

8.14. **mysqlimport** — A Data Import Program

The **mysqlimport** client provides a command-line interface to the `LOAD DATA INFILE` SQL statement. Most options to **mysqlimport** correspond directly to clauses of `LOAD DATA INFILE` syntax. See [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).

Invoke **mysqlimport** like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, **mysqlimport** strips any extension from the filename and uses the result to determine the name of the table into which to import the file's contents. For example, files named `patient.txt`, `patient.text`, and `patient` all would be imported into a table named `patient`.

mysqlimport supports the following options:

- `--help, -?`

Display a help message and exit.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--columns=column_list, -c column_list`

This option takes a comma-separated list of column names as its value. The order of the column names indicates how to match data file columns with table columns.

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--debug[=debug_options], --# [debug_options]`

Write a debugging log. The *debug_options* string often is `'d:t:o,file_name'`.

- `--default-character-set=charset_name`

Use *charset_name* as the default character set. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--delete, -D`

Empty the table before importing the text file.

- `--fields-terminated-by=..., --fields-enclosed-by=..., --fields-optionally-enclosed-by=..., --fields-escaped-by=..., --lines-terminated-by=...`

These options have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. For example, to import Windows files that have lines terminated with carriage return/linefeed pairs, use `--lines-terminated-by="\r\n"`. (You might have to double the backslashes, depending on the escaping conventions of your command interpreter.) See [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).

- `--force, -f`

Ignore errors. For example, if a table for a text file does not exist, continue processing any remaining files. Without `--force`, **mysqlimport** exits if a table does not exist.

- `--host=host_name, -h host_name`

Import data to the MySQL server on the given host. The default host is `localhost`.

- `--ignore, -i`

See the description for the `--replace` option.

- `--ignore-lines=N`

Ignore the first N lines of the data file.

- `--local, -L`

Read input files locally from the client host.

- `--lock-tables, -l`

Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- `--low-priority`

Use `LOW_PRIORITY` when loading the table.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--replace, -r`

The `--replace` and `--ignore` options control handling of input rows that duplicate existing rows on unique key values. If you specify `--replace`, new rows replace existing rows that have the same unique key value. If you specify `--ignore`, input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- `--silent, -s`

Silent mode. Produce output only when errors occur.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.9.7.3, “SSL Command Options”](#).

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does.

- `--version, -V`

Display version information and exit.

Here is a sample session that demonstrates use of **mysqlimport**:

```
shell> mysql -e 'CREATE TABLE impptest(id INT, n VARCHAR(30))' test
shell> ed
a
100      Max Sydow
101      Count Dracula
.
w impptest.txt
32
q
shell> od -c impptest.txt
00000000  1  0  0  \t  M  a  x           S  y  d  o  w  \n  1
00000020  1  \t  C  o  u  n  t           D  r  a  c  u  l  a
00000040
shell> mysqlimport --local test impptest.txt
test.impptest: Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
```

```
shell> mysql -e 'SELECT * FROM imptest' test
```

```
+-----+-----+
| id    | n                |
+-----+-----+
| 100   | Max Sydow       |
| 101   | Count Dracula  |
+-----+-----+
```

8.15. `mysqlshow` — Display Database, Table, and Column Information

The `mysqlshow` client can be used to quickly see which databases exist, their tables, or a table's columns or indexes.

`mysqlshow` provides a command-line interface to several SQL `SHOW` statements. See [Section 13.5.4, “SHOW Syntax”](#). The same information can be obtained by using those statements directly. For example, you can issue them from the `mysql` client program.

Invoke `mysqlshow` like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, a list of database names is shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters (`*`, `?`, `%`, or `_`), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. `*` and `?` characters are converted into SQL `%` and `_` wildcard characters. This might cause some confusion when you try to display the columns for a table with a `_` in the name, because in this case, `mysqlshow` shows you only the table names that match the pattern. This is easily fixed by adding an extra `%` last on the command line as a separate argument.

`mysqlshow` supports the following options:

- `--help, -?`

Display a help message and exit.

- `--character-sets-dir=path`

The directory where character sets are installed. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--count`

Show the number of rows per table. This can be slow for non-MyISAM tables. This option was added in MySQL 5.0.6.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The *debug_options* string often is `'d:t:o,file_name'`.

- `--default-character-set=charset_name`

Use *charset_name* as the default character set. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- `--host=host_name, -h host_name`

Connect to the MySQL server on the given host.

- `--keys, -k`

Show table indexes.

- `--password[=password], -p[password]`

The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

Specifying a password on the command line should be considered insecure. See [Section 5.9.6, “Keeping Your Password Secure”](#).

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

The connection protocol to use.

- `--show-table-type, -t`

Show a column indicating the table type, as in `SHOW FULL TABLES`. The type is `BASE TABLE` or `VIEW`. This option was added in MySQL 5.0.4.

- `--socket=path, -S path`

For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

Options that begin with `--ssl` specify whether to connect to the server via SSL and indicate where to find SSL keys and certificates. See [Section 5.9.7.3, “SSL Command Options”](#).

- `--status, -i`

Display extra information about each table.

- `--user=user_name, -u user_name`

The MySQL username to use when connecting to the server.

- `--verbose, -v`

Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version, -V`

Display version information and exit.

8.16. **mysql_zap** — Kill Processes That Match a Pattern

mysql_zap kills processes that match a pattern. It uses the **ps** command and Unix signals, so it runs on Unix and Unix-like systems.

Invoke **mysql_zap** like this:

```
shell> mysql_zap [-signal] [-?Ift] pattern
```

A process matches if its output line from the **ps** command contains the pattern. By default, **mysql_zap** asks for confirmation for each process. Respond **y** to kill the process, or **q** to exit **mysql_zap**. For any other response, **mysql_zap** does not attempt to kill the process.

If the **-signal** option is given, it specifies the name or number of the signal to send to each process. Otherwise, **mysql_zap** tries first with **TERM** (signal 15) and then with **KILL** (signal 9).

mysql_zap understands the following additional options:

- **--help**, **-?**, **-I**

Display a help message and exit.

- **-f**

Force mode. **mysql_zap** attempts to kill each process without confirmation.

- **-t**

Test mode. Display information about each process but do not kill it.

8.17. perror — Explain Error Codes

For most system errors, MySQL displays, in addition to an internal text message, the system error code in one of the following styles:

```
message ... (errno: #)
message ... (Errcode: #)
```

You can find out what the error code means by examining the documentation for your system or by using the **perror** utility.

perror prints a description for a system error code or for a storage engine (table handler) error code.

Invoke **perror** like this:

```
shell> perror [options] errorcode ...
```

Example:

```
shell> perror 13 64
Error code 13: Permission denied
Error code 64: Machine is not on the network
```

To obtain the error message for a MySQL Cluster error code, invoke **perror** with the `--ndb` option:

```
shell> perror --ndb errorcode
```

Note that the meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

perror supports the following options:

- `--help`, `--info`, `-I`, `-?`

Display a help message and exit.

- `--ndb`

Print the error message for a MySQL Cluster error code.

- `--silent, -s`

Silent mode. Print only the error message.

- `--verbose, -v`

Verbose mode. Print error code and message. This is the default behavior.

- `--version, -V`

Display version information and exit.

8.18. **replace** — A String-Replacement Utility

The **replace** utility program changes strings in place in files or on the standard input.

Invoke **replace** in one of the following ways:

```
shell> replace from to [from to] ... -- file [file] ...  
shell> replace from to [from to] ... < file
```

from represents a string to look for and *to* represents its replacement. There can be one or more pairs of strings.

Use the `--` option to indicate where the string-replacement list ends and the filenames begin. In this case, any file named on the command line is modified in place, so you may want to make a copy of the original before converting it. *replace* prints a message indicating which of the input files it actually modifies.

If the `--` option is not given, **replace** reads the standard input and writes to the standard output.

replace uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps a and b in the given files, `file1` and `file2`:

```
shell> replace a b b a -- file1 file2 ...
```

The **replace** program is used by **msql2mysql**. See [Section 22.9.1, “msql2mysql — Convert mSQL Programs for Use with MySQL”](#).

replace supports the following options:

- `-?`, `-I`

Display a help message and exit.

- `-# debug_options`

Write a debugging log. The `debug_options` string often is

'd:t:o, file_name'.

- -s

Silent mode. Print less information what the program does.

- -v

Verbose mode. Print more information about what the program does.

- -V

Display version information and exit.

Chapter 9. Language Structure

Table of Contents

[9.1. Literal Values](#)

[9.1.1. Strings](#)

[9.1.2. Numbers](#)

[9.1.3. Hexadecimal Values](#)

[9.1.4. Boolean Values](#)

[9.1.5. Bit-Field Values](#)

[9.1.6. NULL Values](#)

[9.2. Database, Table, Index, Column, and Alias Names](#)

[9.2.1. Identifier Qualifiers](#)

[9.2.2. Identifier Case Sensitivity](#)

[9.3. User-Defined Variables](#)

[9.4. Comment Syntax](#)

[9.5. Treatment of Reserved Words in MySQL](#)

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as database, table, and column names
- User-defined and system variables
- Comments
- Reserved words

9.1. Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal values, boolean values, and `NULL`. The section also covers the various nuances and “gotchas” that you may run into when dealing with these basic types in MySQL.

9.1.1. Strings

A string is a sequence of bytes or characters, enclosed within either single quote (‘ ’) or double quote (“ ”) characters. Examples:

```
'a string'  
"another string"
```

If the `ANSI_QUOTES` SQL mode is enabled, string literals can be quoted only within single quotes because a string quoted within double quotes is interpreted as an identifier.

A *binary string* is a string of bytes that has no character set or collation. A *non-binary string* is a string of characters that has a character set and collation. For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte. For non-binary strings the unit is the character and some character sets allow multi-byte characters. Character value ordering is a function of the string collation.

String literals may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';  
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For more information about these forms of string syntax, see [Section 10.3.5, “Character String Literal Character Set and Collation”](#).

Within a string, certain sequences have special meaning. Each of these

sequences begins with a backslash (`\`), known as the *escape character*. MySQL recognizes the following escape sequences:

<code>\0</code>	An ASCII 0 (NUL) character.
<code>\'</code>	A single quote (<code>'</code>) character.
<code>\"</code>	A double quote (<code>"</code>) character.
<code>\b</code>	A backspace character.
<code>\n</code>	A newline (linefeed) character.
<code>\r</code>	A carriage return character.
<code>\t</code>	A tab character.
<code>\Z</code>	ASCII 26 (Control-Z). See note following the table.
<code>\\</code>	A backslash (<code>\</code>) character.
<code>\%</code>	A <code>%</code> character. See note following the table.
<code>_</code>	A <code>_</code> character. See note following the table.

For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, `\x` is just `x`.

These sequences are case sensitive. For example, `\b` is interpreted as a backspace, but `\B` is interpreted as `B`.

The ASCII 26 character can be encoded as `\Z` to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use `mysql db_name < file_name`.

The `\%` and `_` sequences are used to search for literal instances of `%` and `_` in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of the `LIKE` operator in [Section 12.3.1, “String Comparison Functions”](#). If you use `\%` or `_` in non-pattern-matching contexts, they evaluate to the strings `\%` and `_`, not to `%` and `_`.

There are several ways to include quote characters within a string:

- A `'` inside a string quoted with `'` may be written as `''`.
- A `"` inside a string quoted with `"` may be written as `""`.

- Precede the quote character by an escape character (‘\’).
- A ‘\’ inside a string quoted with ‘”’ needs no special treatment and need not be doubled or escaped. In the same way, ‘”’ inside a string quoted with ‘\’ needs no special treatment.

The following SELECT statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel'lo', '\hello';
+-----+-----+-----+-----+-----+
| hello | "hello" | ""hello"" | hel'lo | \hello |
+-----+-----+-----+-----+-----+
```

```
mysql> SELECT "hello", "'hello'", '""hello""', "hel"lo", "\"hello";
+-----+-----+-----+-----+-----+
| hello | 'hello' | '""hello"" | hel"lo | "\"hello |
+-----+-----+-----+-----+-----+
```

```
mysql> SELECT 'This\nIs\nFour\nLines';
+-----+
| This
| Is
| Four
| Lines |
+-----+
```

```
mysql> SELECT 'disappearing\ backslash';
+-----+
| disappearing backslash |
+-----+
```

If you want to insert binary data into a string column (such as a BLOB column), the following characters must be represented by escape sequences:

NULL	NUL byte (ASCII 0). Represent this character by ‘\0’ (a backslash followed by an ASCII ‘0’ character).
\	Backslash (ASCII 92). Represent this character by ‘\\’.
'	Single quote (ASCII 39). Represent this character by ‘\’.
"	Double quote (ASCII 34). Represent this character by ‘\”’.

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in

two ways:

- Process the string with a function that escapes the special characters. In a C program, you can use the `mysql_real_escape_string()` C API function to escape characters. See [Section 22.2.3.52, “mysql_real_escape_string\(\)”](#). The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See [Section 22.4, “MySQL Perl API”](#). Other language interfaces may provide a similar capability.
- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

9.1.2. Numbers

Integers are represented as a sequence of digits. Floats use ‘.’ as a decimal separator. Either type of number may be preceded by ‘-’ or ‘+’ to indicate a negative or positive value, respectively

Examples of valid integers:

```
1221
0
-32
```

Examples of valid floating-point numbers:

```
294.42
-32032.6809e+10
148.00
```

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

9.1.3. Hexadecimal Values

MySQL supports hexadecimal values. In numeric contexts, these act like integers (64-bit precision). In string contexts, these act like binary strings, where each pair of hex digits is converted to a character:

```
mysql> SELECT x'4D7953514C';
      -> 'MySQL'
mysql> SELECT 0xa+0;
      -> 10
mysql> SELECT 0x5061756c;
      -> 'Paul'
```

The default type of a hexadecimal value is a string. If you want to ensure that the value is treated as a number, you can use `CAST(... AS UNSIGNED)`:

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
      -> 'A', 65
```

The `x'hexstring'` syntax is based on standard SQL. The `0x` syntax is based on ODBC. Hexadecimal strings are often used by ODBC to supply values for BLOB columns.

You can convert a string or a number to a string in hexadecimal format with the `HEX()` function:

```
mysql> SELECT HEX('cat');
      -> '636174'
mysql> SELECT 0x636174;
      -> 'cat'
```

9.1.4. Boolean Values

The constants `TRUE` and `FALSE` evaluate to `1` and `0`, respectively. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
      -> 1, 1, 0, 0
```

9.1.5. Bit-Field Values

Beginning with MySQL 5.0.3, bit-field values can be written using `b'value'` notation. *value* is a binary value written using zeros and ones.

Bit-field notation is convenient for specifying values to be assigned to `BIT` columns:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
```

```
mysql> INSERT INTO t SET b = b'1010';
+-----+-----+-----+-----+
| b+0   | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+-----+-----+-----+-----+
| 255   | 11111111 | 377      | FF       |
| 10    | 1010     | 12       | A        |
+-----+-----+-----+-----+
```

9.1.6. NULL Values

The NULL value means “no data.” NULL can be written in any lettercase.

Be aware that the NULL value is different from values such as 0 for numeric types or the empty string for string types. See [Section A.5.3, “Problems with NULL Values”](#).

For text file import or export operations performed with LOAD DATA INFILE or SELECT ... INTO OUTFILE, NULL is represented by the \N sequence. See [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).

9.2. Database, Table, Index, Column, and Alias Names

Database, table, index, column, and alias names are identifiers. This section describes the allowable syntax for identifiers in MySQL.

The following table describes the maximum length for each type of identifier.

Identifier	Maximum Length
Database	64
Table	64
Column	64
Index	64
Alias	255

There are some restrictions on the characters that may appear in identifiers:

- No identifier can contain ASCII 0 (0x00) or a byte with a value of 255.
- The use of identifier quote characters in identifiers is permitted, although it is best to avoid doing so if possible.
- Database, table, and column names should not end with space characters.
- Database names cannot contain '/', '\', '.', or characters that are not allowed in a directory name.
- Table names cannot contain '/', '\', '.', or characters that are not allowed in a filename.

Identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions that are stored in .frm files and to identifiers stored in the grant tables in the mysql database. The sizes of the string columns in the grant tables (and in any other tables) in MySQL 5.0 are given as number of characters. This means that (unlike some earlier versions of MySQL) you can use multi-byte characters without reducing the number of characters allowed for values stored in these columns.

An identifier may be quoted or unquoted. If an identifier is a reserved word or contains special characters, you *must* quote it whenever you refer to it. (Exception: A word that follows a period in a qualified name must be an identifier, so it is not necessary to quote it, even if it is a reserved word.) For a list of reserved words, see [Section 9.5, “Treatment of Reserved Words in MySQL”](#). Special characters are those outside the set of alphanumeric characters from the current character set, ‘_’, and ‘\$’.

The identifier quote character is the backtick (‘`’):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the ANSI_QUOTES SQL mode is enabled, it is also allowable to quote identifiers within double quotes:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax. (...)
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

Note: Because the ANSI_QUOTES mode causes the server to interpret double-quoted strings as identifiers, string literals must be enclosed within single quotes when this mode is enabled. They cannot be enclosed within double quotes.

The server SQL mode is controlled as described in [Section 5.2.5, “The Server SQL Mode”](#).

Identifier quote characters can be included within an identifier *if you quote the identifier*. If the character to be included within the identifier is the same as that used to quote the identifier itself, then you need to double the character. The following statement creates a table named a`b that contains a column named c"d:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

It is recommended that you do not use names of the form Me or MeN, where M and N are integers. For example, avoid using 1e or 2e2 as identifiers, because an expression such as 1e+3 is ambiguous. Depending on context, it might be interpreted as the expression 1e + 3 or as the number 1e+3.

Be careful when using MD5() to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

9.2.1. Identifier Qualifiers

MySQL allows names that consist of a single identifier or multiple identifiers. The components of a multiple-part name should be separated by period (‘.’) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which the final identifier is interpreted.

In MySQL you can refer to a column using any of the following forms:

Column Reference	Meaning
<i>col_name</i>	The column <i>col_name</i> from whichever table used in the statement contains a column of that name.
<i>tbl_name.col_name</i>	The column <i>col_name</i> from table <i>tbl_name</i> of the default database.
<i>db_name.tbl_name.col_name</i>	The column <i>col_name</i> from table <i>tbl_name</i> of the database <i>db_name</i> .

If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write ``my-table`.`my-column``, not ``my-table.my-column``.

You need not specify a *tbl_name* or *db_name.tbl_name* prefix for a column reference in a statement unless the reference would be ambiguous. Suppose that tables *t1* and *t2* each contain a column *c*, and you retrieve *c* in a SELECT statement that uses both *t1* and *t2*. In this case, *c* is ambiguous because it is not unique among the tables used in the statement. You must qualify it with a table name as *t1.c* or *t2.c* to indicate which table you mean. Similarly, to retrieve from a table *t* in database *db1* and from a table *t* in database *db2* in the same statement, you must refer to columns in those tables as *db1.t.col_name* and *db2.t.col_name*.

A word that follows a period in a qualified name must be an identifier, so it is not necessary to quote it, even if it is a reserved word.

The syntax `.tbl_name` means the table `tbl_name` in the default database. This syntax is accepted for ODBC compatibility because some ODBC programs prefix table names with a `'.'` character.

9.2.2. Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage engine). Consequently, the case sensitivity of the underlying operating system determines the case sensitivity of database and table names. This means database and table names are case sensitive in most varieties of Unix, and not case sensitive in Windows. One notable exception is Mac OS X, which is Unix-based but uses a default filesystem type (HFS+) that is not case sensitive. However, Mac OS X also supports UFS volumes, which are case sensitive just as on any Unix. See [Section 1.9.4, “MySQL Extensions to Standard SQL”](#). The `lower_case_table_names` system variable also affects how the server handles identifier case sensitivity, as described later in this section.

Note: Although database and table names are not case sensitive on some platforms, you should not refer to a given database or table using different cases within the same statement. The following statement would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column, index and stored routine names are not case sensitive on any platform, nor are column aliases. Trigger names are case sensitive.

By default, table aliases are case sensitive on Unix, but not so on Windows or Mac OS X. The following statement would not work on Unix, because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a  
-> WHERE a.col_name = 1 OR A.col_name = 2;
```

However, this same statement is permitted on Windows. To avoid problems caused by such differences, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the `lower_case_table_names` system variable, which you can set when starting **mysqld**. `lower_case_table_names` can take the values shown in the following table. On Unix, the default value of `lower_case_table_names` is 0. On Windows the default value is 1. On Mac OS X, the default value is 2.

Value	Meaning
0	Table and database names are stored on disk using the lettercase specified in the <code>CREATE TABLE</code> or <code>CREATE DATABASE</code> statement. Name comparisons are case sensitive. Note that if you force this variable to 0 with <code>--lower-case-table-names=0</code> on a case-insensitive filesystem and access MyISAM table names using different lettercases, index corruption may result.
1	Table names are stored in lowercase on disk and name comparisons are not case sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names and table aliases.
2	Table and database names are stored on disk using the lettercase specified in the <code>CREATE TABLE</code> or <code>CREATE DATABASE</code> statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case sensitive. Note: This works <i>only</i> on filesystems that are not case sensitive! InnoDB table names are stored in lowercase, as for <code>lower_case_table_names=1</code> .

If you are using MySQL on only one platform, you don't normally have to change the `lower_case_table_names` variable. However, you may encounter difficulties if you want to transfer tables between platforms that differ in filesystem case sensitivity. For example, on Unix, you can have two different tables named `my_table` and `MY_TABLE`, but on Windows these two names are considered identical. To avoid data transfer problems stemming from lettercase of database or table names, you have two options:

- Use `lower_case_table_names=1` on all systems. The main disadvantage with this is that when you use `SHOW TABLES` or `SHOW DATABASES`, you don't see the names in their original lettercase.
- Use `lower_case_table_names=0` on Unix and `lower_case_table_names=2` on Windows. This preserves the lettercase of database and table names. The

disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

Exception: If you are using InnoDB tables, you should set `lower_case_table_names` to 1 on all platforms to force names to be converted to lowercase.

Note that if you plan to set the `lower_case_table_names` system variable to 1 on Unix, you must first convert your old database and table names to lowercase before restarting **mysqld** with the new variable setting.

9.3. User-Defined Variables

You can store a value in a user-defined variable and then refer to it later. This enables you to pass values from one statement to another. *User-defined variables are connection-specific*. That is, a user variable defined by one client cannot be seen or used by other clients. All variables for a given client connection are automatically freed when that client exits.

User variables are written as `@var_name`, where the variable name `var_name` may consist of alphanumeric characters from the current character set, `'.'`, `'_'`, and `'$'`. The default character set is `latin1` (cp1252 West European). This may be changed with the `--default-character-set` option to **mysqld**. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#). A user variable name can contain other characters if you quote it as a string or identifier (for example, `@'my-var'`, `@"my-var"`, or `@`my-var``).

Note: User variable names are case sensitive before MySQL 5.0 and not case sensitive in MySQL 5.0 and up.

One way to set a user-defined variable is by issuing a SET statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For SET, either `=` or `:=` can be used as the assignment operator. The `expr` assigned to each variable can evaluate to an integer, real, string, or NULL value. However, if the value of the variable is selected in a result set, it is returned to the client as a string.

You can also assign a value to a user variable in statements other than SET. In this case, the assignment operator must be `:=` and not `=` because `=` is treated as a comparison operator in non-SET statements:

```
mysql> SET @t1=0, @t2=0, @t3=0;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+-----+-----+-----+-----+
| @t1:=(@t2:=1)+@t3:=4 | @t1 | @t2 | @t3 |
+-----+-----+-----+-----+
|                    5 |    5 |    1 |    4 |
+-----+-----+-----+-----+
```

User variables may be used in contexts where expressions are allowed. This does not currently include contexts that explicitly require a literal value, such as in the `LIMIT` clause of a `SELECT` statement, or the `IGNORE N LINES` clause of a `LOAD DATA` statement.

If a user variable is assigned a string value, it has the same character set and collation as the string. The coercibility of user variables is implicit as of MySQL 5.0.3. (This is the same coercibility as for table column values.)

Note: In a `SELECT` statement, each expression is evaluated only when sent to the client. This means that in a `HAVING`, `GROUP BY`, or `ORDER BY` clause, you cannot refer to an expression that involves variables that are set in the `SELECT` list. For example, the following statement does *not* work as expected:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

The reference to `b` in the `HAVING` clause refers to an alias for an expression in the `SELECT` list that uses `@aa`. This does not work as expected: `@aa` contains the value of `id` from the previous selected row, not from the current row.

The order of evaluation for user variables is undefined and may change based on the elements contained within a given query. In `SELECT @a, @a := @a+1 ...`, you might think that MySQL will evaluate `@a` first and then do an assignment second, but changing the query (for example, by adding a `GROUP BY`, `HAVING`, or `ORDER BY` clause) may change the order of evaluation.

The general rule is to never assign a value to a user variable in one part of a statement *and* use the same variable in some other part the same statement. You might get the results you expect, but this is not guaranteed.

Another issue with setting a variable and using it in the same statement is that the default result type of a variable is based on the type of the variable at the start of the statement. The following example illustrates this:

```
mysql> SET @a='test';  
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

For this `SELECT` statement, MySQL reports to the client that column one is a string and converts all accesses of `@a` to strings, even though `@a` is set to a number for the second row. After the `SELECT` statement executes, `@a` is regarded

as a number for the next statement.

To avoid problems with this behavior, either do not set and use the same variable within a single statement, or else set the variable to `0`, `0.0`, or `' '` to define its type before you use it.

If you refer to a variable that has not been initialized, it has a value of `NULL` and a type of string.

9.4. Comment Syntax

MySQL Server supports three comment styles:

- From a '#' character to the end of the line.
- From a '--' sequence to the end of the line. In MySQL, the '--' (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in [Section 1.9.5.7, "--' as the Start of a Comment"](#).
- From a /* sequence to the following */ sequence, as in the C programming language. This syntax allows a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;      # This comment continues to the end of line
mysql> SELECT 1+1;      -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

MySQL Server supports some variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the STRAIGHT_JOIN keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the ‘!’ character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The comment syntax just described applies to how the **mysqld** server parses SQL statements. The **mysql** client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.)

9.5. Treatment of Reserved Words in MySQL

A common problem stems from trying to use an identifier such as a table or column name that is a reserved word such as `SELECT` or the name of a built-in MySQL data type or function such as `TIMESTAMP` or `GROUP`.

If an identifier is a reserved word, you must quote it as described in [Section 9.2, “Database, Table, Index, Column, and Alias Names”](#). Exception: A word that follows a period in a qualified name must be an identifier, so it is not necessary to quote it, even if it is a reserved word.

You are permitted to use function names as identifiers. For example, `ABS` is acceptable as a column name. However, by default, no whitespace is allowed in function invocations between the function name and the following ‘(’ character. This requirement allows a function call to be distinguished from a reference to a column name.

A side effect of this behavior is that omitting a space in some contexts causes an identifier to be interpreted as a function name. For example, this statement is legal:

```
mysql> CREATE TABLE abs (val INT);
```

But omitting the space after `abs` causes a syntax error because the statement then appears to invoke the `ABS()` function:

```
mysql> CREATE TABLE abs(val INT);
ERROR 1064 (42000) at line 2: You have an error in your SQL
syntax ... near 'abs(val INT)'
```

If the `IGNORE_SPACE` SQL mode is enabled, the server allows function invocations to have whitespace between a function name and the following ‘(’ character. This causes function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in [Section 9.2, “Database, Table, Index, Column, and Alias Names”](#). The server SQL mode is controlled as described in [Section 5.2.5, “The Server SQL Mode”](#).

The words in the following table are explicitly reserved in MySQL 5.0. At some

point, you might update to a higher version, so it's a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the words in the table are forbidden by standard SQL as column or table names (for example, GROUP). A few are reserved because MySQL needs them and (currently) uses a **yacc** parser. A reserved word can be used as an identifier if you quote it.

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	BEFORE
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGE	CHAR	CHARACTER
CHECK	COLLATE	COLUMN
CONDITION	CONNECTION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FLOAT4
FLOAT8	FOR	FORCE
FOREIGN	FROM	FULLTEXT

GRANT	GROUP	HAVING
HIGH_PRIORITY	HOUR_MICROSECOND	HOUR_MINUTE
HOUR_SECOND	IF	IGNORE
IN	INDEX	INFILE
INNER	INOUT	INSENSITIVE
INSERT	INT	INT1
INT2	INT3	INT4
INT8	INTEGER	INTERVAL
INTO	IS	ITERATE
JOIN	KEY	KEYS
KILL	LEADING	LEAVE
LEFT	LIKE	LIMIT
LINES	LOAD	LOCALTIME
LOCALTIMESTAMP	LOCK	LONG
LOBLOB	LONGTEXT	LOOP
LOW_PRIORITY	MATCH	MEDIUMBLOB
MEDIUMINT	MEDIUMTEXT	MIDDLEINT
MINUTE_MICROSECOND	MINUTE_SECOND	MOD
MODIFIES	NATURAL	NOT
NO_WRITE_TO_BINLOG	NULL	NUMERIC
ON	OPTIMIZE	OPTION
OPTIONALLY	OR	ORDER
OUT	OUTER	OUTFILE
PRECISION	PRIMARY	PROCEDURE
PURGE	RAID0	READ
READS	REAL	REFERENCES
REGEXP	RELEASE	RENAME
REPEAT	REPLACE	REQUIRE
RESTRICT	RETURN	REVOKE
RIGHT	RLIKE	SCHEMA
SCHEMAS	SECOND_MICROSECOND	SELECT

SENSITIVE	SEPARATOR	SET
SHOW	SMALLINT	SONAME
SPATIAL	SPECIFIC	SQL
SQLEXCEPTION	SQLSTATE	SQLWARNING
SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS	SQL_SMALL_RE
SSL	STARTING	STRAIGHT_JOIN
TABLE	TERMINATED	THEN
TINYBLOB	TINYINT	TINYTEXT
TO	TRAILING	TRIGGER
TRUE	UNDO	UNION
UNIQUE	UNLOCK	UNSIGNED
UPDATE	UPGRADE	USAGE
USE	USING	UTC_DATE
UTC_TIME	UTC_TIMESTAMP	VALUES
VARBINARY	VARCHAR	VARCHARACTERE
VARYING	WHEN	WHERE
WHILE	WITH	WRITE
X509	XOR	YEAR_MONTH
ZEROFILL		

The following are new reserved words in MySQL 5.0: ASENSITIVE, CALL, CONDITION, CONNECTION, CONTINUE, CURSOR, DECLARE, DETERMINISTIC, EACH, ELSEIF, EXIT, FETCH, GOTO, INOUT, INSENSITIVE, ITERATE, LABEL, LEAVE, LOOP, MODIFIES, OUT, READS, RELEASE, REPEAT, RETURN, SCHEMA, SCHEMAS, SENSITIVE, SPECIFIC, SQL, SQLEXCEPTION, SQLSTATE, SQLWARNING, TRIGGER, UNDO, UPGRADE, WHILE.

MySQL allows some keywords to be used as unquoted identifiers because many people previously used them. Examples are those in the following list:

- ACTION
- BIT

- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

Chapter 10. Character Set Support

Table of Contents

- [10.1. Character Sets and Collations in General](#)
- [10.2. Character Sets and Collations in MySQL](#)
- [10.3. Specifying Character Sets and Collations](#)
 - [10.3.1. Server Character Set and Collation](#)
 - [10.3.2. Database Character Set and Collation](#)
 - [10.3.3. Table Character Set and Collation](#)
 - [10.3.4. Column Character Set and Collation](#)
 - [10.3.5. Character String Literal Character Set and Collation](#)
 - [10.3.6. National Character Set](#)
 - [10.3.7. Examples of Character Set and Collation Assignment](#)
 - [10.3.8. Compatibility with Other DBMSs](#)
- [10.4. Connection Character Sets and Collations](#)
- [10.5. Collation Issues](#)
 - [10.5.1. Using COLLATE in SQL Statements](#)
 - [10.5.2. COLLATE Clause Precedence](#)
 - [10.5.3. BINARY Operator](#)
 - [10.5.4. Some Special Cases Where the Collation Determination Is Tricky](#)
 - [10.5.5. Collations Must Be for the Right Character Set](#)
 - [10.5.6. An Example of the Effect of Collation](#)
- [10.6. Operations Affected by Character Set Support](#)
 - [10.6.1. Result Strings](#)
 - [10.6.2. CONVERT\(\) and CAST\(\)](#)
 - [10.6.3. SHOW Statements and INFORMATION_SCHEMA](#)
- [10.7. Unicode Support](#)
- [10.8. UTF-8 for Metadata](#)
- [10.9. Character Sets and Collations That MySQL Supports](#)
 - [10.9.1. Unicode Character Sets](#)
 - [10.9.2. West European Character Sets](#)
 - [10.9.3. Central European Character Sets](#)
 - [10.9.4. South European and Middle East Character Sets](#)
 - [10.9.5. Baltic Character Sets](#)
 - [10.9.6. Cyrillic Character Sets](#)
 - [10.9.7. Asian Character Sets](#)

[10.10. FAQ: MySQL Chinese, Japanese, and Korean Character Sets](#)

[10.10.1. SELECT shows non-Latin characters as "?"s. Why?](#)

[10.10.2. Troubles with GB character sets \(Chinese\)](#)

[10.10.3. Troubles with big5 character set \(Chinese\)](#)

[10.10.4. Troubles with character-set conversions \(Japanese\)](#)

[10.10.5. The Great Yen Sign problem \(Japanese\)](#)

[10.10.6. Troubles with euckr character set \(Korean\)](#)

[10.10.7. The "Data truncated" message](#)

[10.10.8. Troubles with Access, Perl, PHP, etc.](#)

[10.10.9. How can I get old MySQL 4.0 behaviour back?](#)

[10.10.10. Why do some LIKE and FULLTEXT searches fail?](#)

[10.10.11. What CJK character sets are available?](#)

[10.10.12. Is character X available in all character sets?](#)

[10.10.13. Strings don't sort correctly in Unicode \(I\)](#)

[10.10.14. Strings don't sort correctly in Unicode \(II\)](#)

[10.10.15. My supplementary characters get rejected](#)

[10.10.16. Shouldn't it be CJKV \(V for Vietnamese\)?](#)

[10.10.17. Will MySQL fix any CJK problems in version 5.1?](#)

[10.10.18. When will MySQL translate the manual again?](#)

[10.10.19. Whom can I talk to?](#)

MySQL includes character set support that enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. You can specify character sets at the server, database, table, and column level. MySQL supports the use of character sets for the MyISAM, MEMORY, NDBCluster, and InnoDB storage engines.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment
- Syntax for specifying character sets and collations
- Affected functions and operations
- Unicode support
- The character sets and collations that are available, with notes

Character set issues affect data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about character set-related issues in client/server communication, see [Section 10.4, “Connection Character Sets and Collations”](#).

10.1. Character Sets and Collations in General

A *character set* is a set of symbols and encodings. A *collation* is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: 'A', 'B', 'a', 'b'. We give each letter a number: 'A' = 0, 'B' = 1, 'a' = 2, 'b' = 3. The letter 'A' is a symbol, the number 0 is the **encoding** for 'A', and the combination of all four letters and their encodings is a **character set**.

Suppose that we want to compare two string values, 'A' and 'B'. The simplest way to do this is to look at the encodings: 0 for 'A' and 1 for 'B'. Because 0 is less than 1, we say 'A' is less than 'B'. What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): "compare the encodings." We call this simplest of all possible collations a *binary* collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters 'a' and 'b' as equivalent to 'A' and 'B'; (2) then compare the encodings. We call this a *case-insensitive* collation. It's a little more complex than a binary collation.

In real life, most character sets have many characters: not just 'A' and 'B' but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an "accent" is a mark attached to a character as in German 'ö'), and for multiple-character mappings (such as the rule that 'ö' = 'OE' in one of the two German collations).

MySQL can do these things for you:

- Store strings using a variety of character sets
- Compare strings using a variety of collations
- Mix strings with different character sets or collations in the same server, the

same database, or even the same table

- Allow specification of character set and collation at any level

In these respects, MySQL is far ahead of most other database management systems. However, to use these features effectively, you need to know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

10.2. Character Sets and Collations in MySQL

The MySQL server can support multiple character sets. To list the available character sets, use the `SHOW CHARACTER SET` statement. A partial listing follows. For more complete information, see [Section 10.9, “Character Sets and Collations That MySQL Supports”](#).

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation	Max
big5	Big5 Traditional Chinese	big5_chinese_ci	
dec8	DEC West European	dec8_swedish_ci	
cp850	DOS West European	cp850_general_ci	
hp8	HP West European	hp8_english_ci	
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	
latin1	cp1252 West European	latin1_swedish_ci	
latin2	ISO 8859-2 Central European	latin2_general_ci	
swe7	7bit Swedish	swe7_swedish_ci	
ascii	US ASCII	ascii_general_ci	
ujis	EUC-JP Japanese	ujis_japanese_ci	
sjis	Shift-JIS Japanese	sjis_japanese_ci	
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	
tis620	TIS620 Thai	tis620_thai_ci	
euckr	EUC-KR Korean	euckr_korean_ci	
koi8u	KOI8-U Ukrainian	koi8u_general_ci	
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	
greek	ISO 8859-7 Greek	greek_general_ci	
cp1250	Windows Central European	cp1250_general_ci	
gbk	GBK Simplified Chinese	gbk_chinese_ci	
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	
...			

Any given character set always has at least one collation. It may have several collations. To list the collations for a character set, use the `SHOW COLLATION` statement. For example, to see the collations for the `latin1` (cp1252 West European) character set, use this statement to find those collation names that begin with `latin1`:

```
mysql> SHOW COLLATION LIKE 'latin1%';
```

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5			0

latin1_swedish_ci	latin1	8	Yes	Yes	1
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	1
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0

The latin1 collations have the following meanings:

Collation	Meaning
latin1_german1_ci	German DIN-1
latin1_swedish_ci	Swedish/Finnish
latin1_danish_ci	Danish/Norwegian
latin1_german2_ci	German DIN-2
latin1_bin	Binary according to latin1 encoding
latin1_general_ci	Multilingual (Western European)
latin1_general_cs	Multilingual (ISO Western European), case sensitive
latin1_spanish_ci	Modern Spanish

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has one collation that is the *default collation*. For example, the default collation for latin1 is latin1_swedish_ci. The output for SHOW CHARACTER SET indicates which collation is the default for each displayed character set.
- There is a convention for collation names: They start with the name of the character set with which they are associated, they usually include a language name, and they end with _ci (case insensitive), _cs (case sensitive), or _bin (binary).

10.3. Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The following description may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

`CHARACTER SET` is used in clauses that specify a character set. `CHARSET` may be used as a synonym for `CHARACTER SET`.

10.3.1. Server Character Set and Collation

MySQL Server has a server character set and a server collation. These can be set at server startup and changed at runtime.

Initially, the server character set and collation depend on the options that you use when you start **mysqld**. You can use `--character-set-server` for the character set. Along with it, you can add `--collation-server` for the collation. If you don't specify a character set, that is the same as saying `--character-set-server=latin1`. If you specify only a character set (for example, `latin1`) but not a collation, that is the same as saying `--character-set-server=latin1 --collation-server=latin1_swedish_ci` because `latin1_swedish_ci` is the default collation for `latin1`. Therefore, the following three commands all have the same effect:

```
shell> mysqld
shell> mysqld --character-set-server=latin1
shell> mysqld --character-set-server=latin1 \
        --collation-server=latin1_swedish_ci
```

One way to change the settings is by recompiling. If you want to change the default server character set and collation when building from sources, use: `--with-charset` and `--with-collation` as arguments for **configure**. For example:

```
shell> ./configure --with-charset=latin1
```

Or:

```
shell> ./configure --with-charset=latin1 \
```

```
--with-collation=latin1_german1_ci
```

Both **mysqld** and **configure** verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The current server character set and collation can be determined from the values of the `character_set_server` and `collation_server` system variables. These variables can be changed at runtime.

10.3.2. Database Character Set and Collation

Every database has a database character set and a database collation. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

```
ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

The keyword `SCHEMA` can be used instead of `DATABASE`.

All database options are stored in a text file named `db.opt` that can be found in the database directory.

The `CHARACTER SET` and `COLLATE` clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Example:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_
```

MySQL chooses the database character set and database collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` were specified, then character set `X` and collation `Y`.
- If `CHARACTER SET X` was specified without `COLLATE`, then character set `X`

and its default collation.

- If `COLLATE Y` was specified without `CHARACTER SET`, then the character set associated with `Y` and collation `Y`.
- Otherwise, the server character set and server collation.

The database character set and collation are used as default values if the table character set and collation are not specified in `CREATE TABLE` statements. They have no other purpose.

The character set and collation for the default database can be determined from the values of the `character_set_database` and `collation_database` system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level system variables, `character_set_server` and `collation_server`.

10.3.3. Table Character Set and Collation

Every table has a table character set and a table collation. The `CREATE TABLE` and `ALTER TABLE` statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)
    [[DEFAULT] CHARACTER SET charset_name] [COLLATE collation_name]]

ALTER TABLE tbl_name
    [[DEFAULT] CHARACTER SET charset_name] [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... ) CHARACTER SET latin1 COLLATE latin1_danish_c
```

MySQL chooses the table character set and collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` were specified, then character set `X` and collation `Y`.
- If `CHARACTER SET X` was specified without `COLLATE`, then character set `X` and its default collation.

- If `COLLATE Y` was specified without `CHARACTER SET`, then the character set associated with `Y` and collation `Y`.
- Otherwise, the database character set and collation.

The table character set and collation are used as default values if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

10.3.4. Column Character Set and Collation

Every “character” column (that is, a column of type `CHAR`, `VARCHAR`, or `TEXT`) has a column character set and a column collation. Column definition syntax has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
          [CHARACTER SET charset_name] [COLLATE collation_name]
```

Example:

```
CREATE TABLE Table1
(
    column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1_german1_c
);
```

MySQL chooses the column character set and collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` were specified, then character set `X` and collation `Y` are used.
- If `CHARACTER SET X` was specified without `COLLATE`, then character set `X` and its default collation are used.
- If `COLLATE Y` was specified without `CHARACTER SET`, then the character set associated with `Y` and collation `Y`.
- Otherwise, the table character set and collation are used.

The `CHARACTER SET` and `COLLATE` clauses are standard SQL.

10.3.5. Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

A character string literal may have an optional character set introducer and COLLATE clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT 'string';  
SELECT _latin1'string';  
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For the simple statement SELECT 'string', the string has the character set and collation defined by the *character_set_connection* and *collation_connection* system variables.

The *_charset_name* expression is formally called an *introducer*. It tells the parser, “the string that is about to follow uses character set *x*.” Because this has confused people in the past, we emphasize that an introducer does not cause any conversion; it is strictly a signal that does not change the string's value. An introducer is also legal before standard hex literal and numeric hex literal notation (*x*'literal' and 0xnnnn)>.

Examples:

```
SELECT _latin1 x'AABBCC';  
SELECT _latin1 0xAABBCC;
```

MySQL determines a literal's character set and collation in the following manner:

- If both *_X* and COLLATE *Y* were specified, then character set *X* and collation *Y* are used.
- If *_X* is specified but COLLATE is not specified, then character set *X* and its default collation are used.
- Otherwise, the character set and collation given by the

character_set_connection and collation_connection system variables are used.

Examples:

- A string with latin1 character set and latin1_german1_ci collation:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- A string with latin1 character set and its default collation (that is, latin1_swedish_ci):

```
SELECT _latin1'Müller';
```

- A string with the connection default character set and collation:

```
SELECT 'Müller';
```

Character set introducers and the COLLATE clause are implemented according to standard SQL specifications.

10.3.6. National Character Set

Standard SQL defines NCHAR or NATIONAL CHAR as a way to indicate that a CHAR column should use some predefined character set. MySQL 5.0 uses utf8 as this predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8  
NATIONAL CHARACTER(10)  
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8  
NATIONAL VARCHAR(10)  
NCHAR VARCHAR(10)  
NATIONAL CHARACTER VARYING(10)  
NATIONAL CHAR VARYING(10)
```

You can use N'literal' to create a string in the national character set. These two statements are equivalent:

```
SELECT N'some text';
SELECT _utf8'some text';
```

For information on upgrading character sets to MySQL 5.0 from versions prior to 4.1, see the *MySQL 3.23, 4.0, 4.1 Reference Manual*.

10.3.7. Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that's straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this

circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1
  DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
  c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

10.3.8. Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

10.4. Connection Character Sets and Collations

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The server character set and collation can be determined from the values of the `character_set_server` and `collation_server` system variables.
- The character set and collation of the default database can be determined from the values of the `character_set_database` and `collation_database` system variables.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has connection-related character set and collation system variables.

Consider what a “connection” is: It's what you make when you connect to the server. The client sends SQL statements, such as queries, over the connection to the server. The server sends responses, such as result sets, over the connection back to the client. This leads to several questions about character set and collation handling for client connections, each of which can be answered in terms of system variables:

- What character set is the statement in when it leaves the client?

The server takes the `character_set_client` system variable to be the character set in which statements are sent by the client.

- What character set should the server translate a statement to after receiving it?

For this, the server uses the `character_set_connection` and `collation_connection` system variables. It converts statements sent by the client from `character_set_client` to `character_set_connection` (except for string literals that have an introducer such as `_latin1` or `_utf8`). `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` does not matter because columns have their own collation, which has a higher

collation precedence.

- What character set should the server translate to before shipping result sets or error messages back to the client?

The `character_set_results` system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, and result metadata such as column names.

You can fine-tune the settings for these variables, or you can depend on the defaults (in which case, you can skip the rest of this section).

There are two statements that affect the connection character sets:

```
SET NAMES 'charset_name'  
SET CHARACTER SET charset_name
```

`SET NAMES` indicates what character set the client will use to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server “future incoming messages from this client are in character set cp1251.” It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement.)

A `SET NAMES 'x'` statement is equivalent to these three statements:

```
SET character_set_client = x;  
SET character_set_results = x;  
SET character_set_connection = x;
```

Setting `character_set_connection` to `x` also sets `collation_connection` to the default collation for `x`. To specify one of the character set's collations explicitly, use the optional `COLLATE` clause:

```
SET NAMES 'charset_name' COLLATE 'collation_name'
```

`SET CHARACTER SET` is similar to `SET NAMES` but sets the connection character set and collation to be those of the default database. A `SET CHARACTER SET x` statement is equivalent to these three statements:

```
SET character_set_client = x;  
SET character_set_results = x;
```

```
SET collation_connection = @@collation_database;
```

Setting `collation_connection` also sets `character_set_connection` to the character set associated with the collation.

When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

With the **mysql** client, it is not necessary to execute `SET NAMES` every time you start up if you want to use a character set different from the default. You can add the `--default-character-set` option setting to your **mysql** statement line, or in your option file. For example, the following option file setting changes the three character set variables set to `koi8r` each time you invoke **mysql**:

```
[mysql]
default-character-set=koi8r
```

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say `SET NAMES` or `SET CHARACTER SET`, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say `SET NAMES 'latin1'` or `SET CHARACTER SET latin1` before issuing the `SELECT` statement, the server converts the `latin2` values to `latin1` just before sending results back. Conversion may be lossy if there are characters that are not in both character sets.

If you do not want the server to perform any conversion of result sets, set `character_set_results` to `NULL`:

```
SET character_set_results = NULL;
```

Note: Currently, UCS-2 cannot be used as a client character set, which means that `SET NAMES 'ucs2'` does not work.

To see the values of the character set and collation system variables that apply to your connection, use these statements:

```
SHOW VARIABLES LIKE 'character_set%';
SHOW VARIABLES LIKE 'collation%';
```

10.5. Collation Issues

The following sections various aspects of character set collations.

10.5.1. Using COLLATE in SQL Statements

With the COLLATE clause, you can override whatever the default collation is for a comparison. COLLATE may be used in various parts of SQL statements. Here are some examples:

- With ORDER BY:

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- With AS:

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- With GROUP BY:

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- With aggregate functions:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- With DISTINCT:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- With WHERE:

```
SELECT *
FROM t1
```

```

WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;

SELECT *
FROM t1
WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;

```

- With HAVING:

```

SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;

```

10.5.2. COLLATE Clause Precedence

The COLLATE clause has high precedence (higher than ||), so the following two expressions are equivalent:

```

x || y COLLATE z
x || (y COLLATE z)

```

10.5.3. BINARY Operator

The BINARY operator casts the string following it to a binary string. This is an easy way to force a comparison to be done byte by byte rather than character by character. BINARY also causes trailing spaces to be significant.

```

mysql> SELECT 'a' = 'A';
-> 1
mysql> SELECT BINARY 'a' = 'A';
-> 0
mysql> SELECT 'a' = 'a ';
-> 1
mysql> SELECT BINARY 'a' = 'a ';
-> 0

```

BINARY str is shorthand for CAST(str AS BINARY).

The BINARY attribute in character column definitions has a different effect. A character column defined with the BINARY attribute is assigned the binary collation of the column's character set. Every character set has a binary collation. For example, the binary collation for the latin1 character set is latin1_bin, so if the table default character set is latin1, these two column definitions are

equivalent:

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin
```

The effect of BINARY as a column attribute differs from its effect prior to MySQL 4.1. Formerly, BINARY resulted in a column that was treated as a binary string. A binary string is a string of bytes that has no character set or collation, which differs from a non-binary character string that has a binary collation. For both types of strings, comparisons are based on the numeric values of the string unit, but for non-binary strings the unit is the character and some character sets allow multi-byte characters. [Section 11.4.2, “The BINARY and VARBINARY Types”](#).

The use of CHARACTER SET binary in the definition of a CHAR, VARCHAR, or TEXT column causes the column to be treated as a binary data type. For example, the following pairs of definitions are equivalent:

```
CHAR(10) CHARACTER SET binary
BINARY(10)
```

```
VARCHAR(10) CHARACTER SET binary
VARBINARY(10)
```

```
TEXT CHARACTER SET binary
BLOB
```

10.5.4. Some Special Cases Where the Collation Determination Is Tricky

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column x:

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, when multiple operands are involved, there can be ambiguity. For example:

```
SELECT x FROM T WHERE x = 'Y';
```

Should this query use the collation of the column `x`, or of the string literal `'Y'`?

Standard SQL resolves such questions using what used to be called “coercibility” rules. Basically, this means: Both `x` and `'Y'` have collations, so which collation takes precedence? This can be difficult to resolve, but the following rules cover most situations:

- An explicit `COLLATE` clause has a coercibility of 0. (Not coercible at all.)
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column or a stored routine parameter or local variable has a coercibility of 2.
- A “system constant” (the string returned by functions such as `USER()` or `VERSION()`) has a coercibility of 3.
- A literal's collation has a coercibility of 4.
- `NULL` or an expression that is derived from `NULL` has a coercibility of 5.

The preceding coercibility values are current as of MySQL 5.0.3. In MySQL 5.0 prior to 5.0.3, there is no system constant or ignorable coercibility. Functions such as `USER()` have a coercibility of 2 rather than 3, and literals have a coercibility of 3 rather than 4.

Those rules resolve ambiguities in the following manner:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then it is an error if the collations aren't the same.

Examples:

<code>column1 = 'A'</code>	Use collation of <code>column1</code>
<code>column1 = 'A' COLLATE x</code>	Use collation of <code>'A' COLLATE x</code>
<code>column1 COLLATE x = 'A' COLLATE y</code>	Error

The `COERCIBILITY()` function can be used to determine the coercibility of a string expression:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
-> 0
mysql> SELECT COERCIBILITY(VERSION());
-> 3
mysql> SELECT COERCIBILITY('A');
-> 4
```

See [Section 12.9.3, “Information Functions”](#).

10.5.5. Collations Must Be for the Right Character Set

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the `latin2_bin` collation is not legal with the `latin1` character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

10.5.6. An Example of the Effect of Collation

Suppose that column `X` in table `T` has these `latin1` column values:

```
Muffler
Müller
MX Systems
MySQL
```

Suppose also that the column values are retrieved using the following statement:

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

The following table shows the resulting order of the values if we use `ORDER BY` with different collations:

<code>latin1_swedish_ci</code>	<code>latin1_german1_ci</code>	<code>latin1_german2_ci</code>
Muffler	Muffler	Müller
MX Systems	Müller	Muffler

Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

The character that causes the different sort orders in this example is the U with two dots over it (ü), which the Germans call “U-umlaut.”

- The first column shows the result of the `SELECT` using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.
- The second column shows the result of the `SELECT` using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the `SELECT` using the German DIN-2 rule, which says that U-umlaut sorts with UE.

10.6. Operations Affected by Character Set Support

This section describes operations that take character set information into account.

10.6.1. Result Strings

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, `UPPER(x)` returns a string whose character string and collation are the same as that of `x`. The same applies for `INSTR()`, `LCASE()`, `LOWER()`, `LTRIM()`, `MID()`, `REPEAT()`, `REPLACE()`, `REVERSE()`, `RIGHT()`, `RPAD()`, `RTRIM()`, `SOUNDEX()`, `SUBSTRING()`, `TRIM()`, `UCASE()`, and `UPPER()`.

Note: The `REPLACE()` function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has no character set or collation. This can be checked by using the `CHARSET()` and `COLLATION()` functions, both of which return `binary` to indicate that their argument is a binary string:

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+-----+-----+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+-----+-----+
| binary              | binary                  |
+-----+-----+
```

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

- If an explicit `COLLATE x` occurs, use `x`.
- If explicit `COLLATE x` and `COLLATE y` occur, raise an error.

- Otherwise, if all collations are *x*, use *x*.
- Otherwise, the result has no collation.

For example, with `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, the resulting collation is *x*. The same applies for `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()`, and `LEAST()`.

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the `character_set_connection` and `collation_connection` system variables. This applies to `CAST()`, `CONV()`, `FORMAT()`, `HEX()`, `SPACE()`. Before MySQL 5.0.15, it also applies to `CHAR()`.

10.6.2. CONVERT() and CAST()

`CONVERT()` provides a way to convert data between different character sets. The syntax is:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
    SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` is implemented according to the standard SQL specification.

You may also use `CAST()` to convert a string to a different character set. The syntax is:

```
CAST(character_string AS character_data_type CHARACTER SET charset_n)
```

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

If you use `CAST()` without specifying `CHARACTER SET`, the resulting character set and collation are defined by the `character_set_connection` and `collation_connection` system variables. If you use `CAST()` with `CHARACTER SET X`, the resulting character set and collation are `X` and the default collation of `X`.

You may not use a `COLLATE` clause inside a `CAST()`, but you may use it outside. That is, `CAST(... COLLATE ...)` is illegal, but `CAST(...) COLLATE ...` is legal.

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_b
```

10.6.3. SHOW Statements and INFORMATION_SCHEMA

Several `SHOW` statements provide additional character set information. These include `SHOW CHARACTER SET`, `SHOW COLLATION`, `SHOW CREATE DATABASE`, `SHOW CREATE TABLE` and `SHOW COLUMNS`. These statements are described here briefly. For more information, see [Section 13.5.4, “SHOW Syntax”](#).

`INFORMATION_SCHEMA` has several tables that contain information similar to that displayed by the `SHOW` statements. For example, the `CHARACTER_SETS` and `COLLATIONS` tables contain the information displayed by `SHOW CHARACTER SET` and `SHOW COLLATION`. [Chapter 20, The INFORMATION_SCHEMA Database](#).

The `SHOW CHARACTER SET` command shows all available character sets. It takes an optional `LIKE` clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description                | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European      | latin1_swedish_ci | 1      |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1      |
| latin5  | ISO 8859-9 Turkish        | latin5_turkish_ci | 1      |
| latin7  | ISO 8859-13 Baltic         | latin7_general_ci | 1      |
+-----+-----+-----+-----+
```

The output from `SHOW COLLATION` includes all available character sets. It takes an optional `LIKE` clause that indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
```

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5			0
latin1_swedish_ci	latin1	8	Yes	Yes	0
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	0
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0

SHOW CREATE DATABASE displays the CREATE DATABASE statement that creates a given database:

```
mysql> SHOW CREATE DATABASE test;
```

Database	Create Database
test	CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET l

If no COLLATE clause is shown, the default collation for the character set applies.

SHOW CREATE TABLE is similar, but displays the CREATE TABLE statement to create a given table. The column definitions indicate any character set specifications, and the table options include character set information.

The SHOW COLUMNS statement displays the collations of a table's columns when invoked as SHOW FULL COLUMNS. Columns with CHAR, VARCHAR, or TEXT data types have collations. Numeric and other non-character types have no collation (indicated by NULL as the Collation value). For example:

```
mysql> SHOW FULL COLUMNS FROM person\G
```

```
***** 1. row *****
Field: id
Type: smallint(5) unsigned
Collation: NULL
Null: NO
Key: PRI
Default: NULL
Extra: auto_increment
Privileges: select,insert,update,references
Comment:
```

```
***** 2. row *****
Field: name
Type: char(60)
Collation: latin1_swedish_ci
Null: NO
Key:
Default:
Extra:
Privileges: select,insert,update,references
Comment:
```

The character set is not part of the display but is implied by the collation name.

10.7. Unicode Support

MySQL 5.0 supports two character sets for storing Unicode data:

- `ucs2`, the UCS-2 Unicode character set.
- `utf8`, the UTF-8 encoding of the Unicode character set.

In UCS-2 (binary Unicode representation), every character is represented by a two-byte Unicode code with the most significant byte first. For example: LATIN CAPITAL LETTER A has the code `0x0041` and it is stored as a two-byte sequence: `0x00 0x41`. CYRILLIC SMALL LETTER YERU (Unicode `0x044B`) is stored as a two-byte sequence: `0x04 0x4B`. For Unicode characters and their codes, please refer to the [Unicode Home Page](#).

Currently, UCS-2 cannot be used as a client character set, which means that `SET NAMES 'ucs2'` does not work.

The UTF-8 character set (transform Unicode representation) is an alternative way to store Unicode data. It is implemented according to RFC 3629. The idea of the UTF-8 character set is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a two-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use three-byte sequences.

RFC 3629 describes encoding sequences that take from one to four bytes. Currently, MySQL support for UTF-8 does not include four-byte sequences. (An older standard for UTF-8 encoding is given by RFC 2279, which describes UTF-8 sequences that take from one to six bytes. RFC 3629 renders RFC 2279 obsolete; for this reason, sequences with five and six bytes are no longer used.)

Tip: To save space with UTF-8, use `VARCHAR` instead of `CHAR`. Otherwise,

MySQL must reserve three bytes for each character in a CHAR CHARACTER SET utf8 column because that is the maximum possible length. For example, MySQL must reserve 30 bytes for a CHAR(10) CHARACTER SET utf8 column.

10.8. UTF-8 for Metadata

Metadata is “the data about the data.” Anything that *describes* the database — as opposed to being the *contents* of the database — is metadata. Thus column names, database names, usernames, version names, and most of the string results from `SHOW` are metadata. This is also true of the contents of tables in `INFORMATION_SCHEMA`, because those tables by definition contain information about database objects.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, neither the `SHOW` commands nor `SELECT` statements for tables in `INFORMATION_SCHEMA` would work properly because different rows in the same column of the results of these operations would be in different character sets.
- Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the `USER()`, `CURRENT_USER()`, `SESSION_USER()`, `SYSTEM_USER()`, `DATABASE()`, and `VERSION()` functions have the UTF-8 character set by default.

The server sets the `character_set_system` system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| character_set_system | utf8  |
+-----+-----+
```

Storage of metadata using Unicode does *not* mean that the server returns headers

of columns and the results of DESCRIBE functions in the `character_set_system` character set by default. When you use `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set determined by the value of the `character_set_results` system variable, which has a default value of `latin1`. If you want the server to pass metadata results back in a different character set, use the `SET NAMES` statement to force the server to perform character set conversion. `SET NAMES` sets the `character_set_results` and other related system variables. (See [Section 10.4, “Connection Character Sets and Collations”](#).) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client perform the conversion, but this option is not always available for all clients.

If `character_set_results` is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system`).

Error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the `USER()` function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF-8 before the comparison.

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` are automatically converted to `latin1` before the assignment. Automatic conversion is not fully implemented yet, but should work correctly in a later version.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings.

10.9. Character Sets and Collations That MySQL Supports

MySQL supports 70+ collations for 30+ character sets. This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the allowable collations are listed.

You can always list the available character sets and their default collations with the `SHOW CHARACTER SET` statement:

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation
big5	Big5 Traditional Chinese	big5_chinese_ci
dec8	DEC West European	dec8_swedish_ci
cp850	DOS West European	cp850_general_ci
hp8	HP West European	hp8_english_ci
koi8r	KOI8-R Relcom Russian	koi8r_general_ci
latin1	cp1252 West European	latin1_swedish_ci
latin2	ISO 8859-2 Central European	latin2_general_ci
swe7	7bit Swedish	swe7_swedish_ci
ascii	US ASCII	ascii_general_ci
ujis	EUC-JP Japanese	ujis_japanese_ci
sjis	Shift-JIS Japanese	sjis_japanese_ci
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci
tis620	TIS620 Thai	tis620_thai_ci
euckr	EUC-KR Korean	euckr_korean_ci
koi8u	KOI8-U Ukrainian	koi8u_general_ci
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci
greek	ISO 8859-7 Greek	greek_general_ci
cp1250	Windows Central European	cp1250_general_ci
gbk	GBK Simplified Chinese	gbk_chinese_ci
latin5	ISO 8859-9 Turkish	latin5_turkish_ci
armscii8	ARMSCII-8 Armenian	armscii8_general_ci
utf8	UTF-8 Unicode	utf8_general_ci
ucs2	UCS-2 Unicode	ucs2_general_ci
cp866	DOS Russian	cp866_general_ci
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci
macce	Mac Central European	macce_general_ci
macroman	Mac West European	macroman_general_ci
cp852	DOS Central European	cp852_general_ci
latin7	ISO 8859-13 Baltic	latin7_general_ci
cp1251	Windows Cyrillic	cp1251_general_ci
cp1256	Windows Arabic	cp1256_general_ci

cp1257	Windows Baltic	cp1257_general_ci
binary	Binary pseudo charset	binary
geostd8	GEOSTD8 Georgian	geostd8_general_ci
cp932	SJIS for Windows Japanese	cp932_japanese_ci
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci

10.9.1. Unicode Character Sets

MySQL has two Unicode character sets. You can store text in about 650 languages using these character sets.

- ucs2 (UCS-2 Unicode) collations:
 - ucs2_bin
 - ucs2_czech_ci
 - ucs2_danish_ci
 - ucs2_esperanto_ci
 - ucs2_estonian_ci
 - ucs2_general_ci (default)
 - ucs2_hungarian_ci
 - ucs2_icelandic_ci
 - ucs2_latvian_ci
 - ucs2_lithuanian_ci
 - ucs2_persian_ci
 - ucs2_polish_ci
 - ucs2_roman_ci
 - ucs2_romanian_ci

- ucs2_slovak_ci
- ucs2_slovenian_ci
- ucs2_spanish2_ci
- ucs2_spanish_ci
- ucs2_swedish_ci
- ucs2_turkish_ci
- ucs2_unicode_ci
- utf8 (UTF-8 Unicode) collations:
 - utf8_bin
 - utf8_czech_ci
 - utf8_danish_ci
 - utf8_esperanto_ci
 - utf8_estonian_ci
 - utf8_general_ci (default)
 - utf8_hungarian_ci
 - utf8_icelandic_ci
 - utf8_latvian_ci
 - utf8_lithuanian_ci
 - utf8_persian_ci
 - utf8_polish_ci
 - utf8_roman_ci

- `utf8_romanian_ci`
- `utf8_slovak_ci`
- `utf8_slovenian_ci`
- `utf8_spanish2_ci`
- `utf8_spanish_ci`
- `utf8_swedish_ci`
- `utf8_turkish_ci`
- `utf8_unicode_ci`

Note that in the `ucs2_roman_ci` and `utf8_roman_ci` collations, I and J compare as equals, and U and V compare as equals.

The `ucs2_esperanto_ci` and `utf8_esperanto_ci` collations were added in MySQL 5.0.13. The `ucs2_hungarian_ci` and `utf8_hungarian_ci` collations were added in MySQL 5.0.19.

MySQL implements the `utf8_unicode_ci` collation according to the Unicode Collation Algorithm (UCA) described at <http://www.unicode.org/reports/tr10/>. The collation uses the version-4.0.0 UCA weight keys: <http://www.unicode.org/Public/UC4.0.0/allkeys-4.0.0.txt>. The following discussion uses `utf8_unicode_ci`, but it is also true for `ucs2_unicode_ci`.

Currently, the `utf8_unicode_ci` collation has only partial support for the Unicode Collation Algorithm. Some characters are not supported yet. Also, combining marks are not fully supported. This affects primarily Vietnamese and some minority languages in Russia such as Udmurt, Tatar, Bashkir, and Mari.

The most significant feature in `utf8_unicode_ci` is that it supports expansions; that is, when one character compares as equal to combinations of other characters. For example, in German and some other languages ‘ß’ is equal to ‘ss’.

`utf8_general_ci` is a legacy collation that does not support expansions. It can

make only one-to-one comparisons between characters. This means that comparisons for the `utf8_general_ci` collation are faster, but slightly less correct, than comparisons for `utf8_unicode_ci`.

For example, the following equalities hold in both `utf8_general_ci` and `utf8_unicode_ci`:

Ä = A
Ö = O
Ü = U

A difference between the collations is that this is true for `utf8_general_ci`:

ß = s

Whereas this is true for `utf8_unicode_ci`:

ß = ss

MySQL implements language-specific collations for the `utf8` character set only if the ordering with `utf8_unicode_ci` does not work well for a language. For example, `utf8_unicode_ci` works fine for German and French, so there is no need to create special `utf8` collations for these two languages.

`utf8_general_ci` also is satisfactory for both German and French, except that 'ß' is equal to 's', and not to 'ss'. If this is acceptable for your application, then you should use `utf8_general_ci` because it is faster. Otherwise, use `utf8_unicode_ci` because it is more accurate.

`utf8_swedish_ci`, like other `utf8` language-specific collations, is derived from `utf8_unicode_ci` with additional language rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

Ü = Y < Ö

The `utf8_spanish_ci` and `utf8_spanish2_ci` collations correspond to modern Spanish and traditional Spanish, respectively. In both collations, 'ñ' (n-tilde) is a separate letter between 'n' and 'o'. In addition, for traditional Spanish, 'ch' is a separate letter between 'c' and 'd', and 'll' is a separate letter between 'l' and 'm'

10.9.2. West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- `ascii` (US ASCII) collations:
 - `ascii_bin`
 - `ascii_general_ci` (default)
- `cp850` (DOS West European) collations:
 - `cp850_bin`
 - `cp850_general_ci` (default)
- `dec8` (DEC Western European) collations:
 - `dec8_bin`
 - `dec8_swedish_ci` (default)
- `hp8` (HP Western European) collations:
 - `hp8_bin`
 - `hp8_english_ci` (default)
- `latin1` (cp1252 West European) collations:
 - `latin1_bin`
 - `latin1_danish_ci`
 - `latin1_general_ci`
 - `latin1_general_cs`

- `latin1_german1_ci`
- `latin1_german2_ci`
- `latin1_spanish_ci`
- `latin1_swedish_ci` (default)

`latin1` is the default character set. MySQL's `latin1` is the same as the Windows `cp1252` character set. This means it is the same as the official ISO 8859-1 or IANA (Internet Assigned Numbers Authority) `latin1`, but IANA `latin1` treats the code points between `0x80` and `0x9f` as “undefined,” whereas `cp1252`, and therefore MySQL's `latin1`, assign characters for those positions. For example, `0x80` is the Euro sign. For the “undefined” entries in `cp1252`, MySQL translates `0x81` to Unicode `0x0081`, `0x8d` to `0x008d`, `0x8f` to `0x008f`, `0x90` to `0x0090`, and `0x9d` to `0x009d`.

The `latin1_swedish_ci` collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.

The `latin1_german1_ci` and `latin1_german2_ci` collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the “dictionary collation” and DIN-2 is called the “phone book collation.”

- `latin1_german1_ci` (dictionary) rules:

`Ä = A`
`Ö = O`
`Ü = U`
`ß = s`

- `latin1_german2_ci` (phone-book) rules:

`Ä = AE`
`Ö = OE`
`Ü = UE`
`ß = ss`

In the `latin1_spanish_ci` collation, ‘ñ’ (n-tilde) is a separate letter

between 'n' and 'o'.

- macroman (Mac West European) collations:
 - macroman_bin
 - macroman_general_ci (default)
- swe7 (7bit Swedish) collations:
 - swe7_bin
 - swe7_swedish_ci (default)

10.9.3. Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, and Poland.

- cp1250 (Windows Central European) collations:
 - cp1250_bin
 - cp1250_croatian_ci
 - cp1250_czech_cs
 - cp1250_general_ci (default)
- cp852 (DOS Central European) collations:
 - cp852_bin
 - cp852_general_ci (default)
- keybcs2 (DOS Kamenicky Czech-Slovak) collations:
 - keybcs2_bin
 - keybcs2_general_ci (default)

- latin2 (ISO 8859-2 Central European) collations:
 - latin2_bin
 - latin2_croatian_ci
 - latin2_czech_cs
 - latin2_general_ci (default)
 - latin2_hungarian_ci
- macce (Mac Central European) collations:
 - macce_bin
 - macce_general_ci (default)

10.9.4. South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

- armSCII8 (ARMSCII-8 Armenian) collations:
 - armSCII8_bin
 - armSCII8_general_ci (default)
- cp1256 (Windows Arabic) collations:
 - cp1256_bin
 - cp1256_general_ci (default)
- geostd8 (GEOSTD8 Georgian) collations:
 - geostd8_bin
 - geostd8_general_ci (default)

- greek (ISO 8859-7 Greek) collations:
 - greek_bin
 - greek_general_ci (default)
- hebrew (ISO 8859-8 Hebrew) collations:
 - hebrew_bin
 - hebrew_general_ci (default)
- latin5 (ISO 8859-9 Turkish) collations:
 - latin5_bin
 - latin5_turkish_ci (default)

10.9.5. Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- cp1257 (Windows Baltic) collations:
 - cp1257_bin
 - cp1257_general_ci (default)
 - cp1257_lithuanian_ci
- latin7 (ISO 8859-13 Baltic) collations:
 - latin7_bin
 - latin7_estonian_cs
 - latin7_general_ci (default)
 - latin7_general_cs

10.9.6. Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, and Ukrainian languages.

- cp1251 (Windows Cyrillic) collations:
 - cp1251_bin
 - cp1251_bulgarian_ci
 - cp1251_general_ci (default)
 - cp1251_general_cs
 - cp1251_ukrainian_ci
- cp866 (DOS Russian) collations:
 - cp866_bin
 - cp866_general_ci (default)
- koi8r (KOI8-R Relcom Russian) collations:
 - koi8r_bin
 - koi8r_general_ci (default)
- koi8u (KOI8-U Ukrainian) collations:
 - koi8u_bin
 - koi8u_general_ci (default)

10.9.7. Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See [Section 10.9.7.1, “The cp932 Character Set”](#), for additional information about the cp932 and sjis character sets.

- big5 (Big5 Traditional Chinese) collations:

- big5_bin
- big5_chinese_ci (default)
- cp932 (SJIS for Windows Japanese) collations:
 - cp932_bin
 - cp932_japanese_ci (default)
- eucjpms (UJIS for Windows Japanese) collations:
 - eucjpms_bin
 - eucjpms_japanese_ci (default)
- euckr (EUC-KR Korean) collations:
 - euckr_bin
 - euckr_korean_ci (default)
- gb2312 (GB2312 Simplified Chinese) collations:
 - gb2312_bin
 - gb2312_chinese_ci (default)
- gbk (GBK Simplified Chinese) collations:
 - gbk_bin
 - gbk_chinese_ci (default)
- sjis (Shift-JIS Japanese) collations:
 - sjis_bin
 - sjis_japanese_ci (default)
- tis620 (TIS620 Thai) collations:

- `tis620_bin`
- `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:
 - `ujis_bin`
 - `ujis_japanese_ci` (default)

10.9.7.1. The `cp932` Character Set

Why is `cp932` needed?

In MySQL, the `sjis` character set corresponds to the `Shift_JIS` character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See <http://www.iana.org/assignments/character-sets>.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to `Shift_JIS` that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of `Shift_JIS` and its exact name is `Microsoft Windows Codepage : 932` or `cp932`. In addition to the characters supported by `Shift_JIS`, `cp932` supports extension characters such as NEC special characters, NEC selected — IBM extended characters, and IBM extended characters.

Many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted via Unicode (`ucs2`).
- The `sjis` character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the

conversion rule. MySQL supports only one of these rules (described later).

The MySQL cp932 character set is designed to solve these problems. It is available as of MySQL 5.0.3.

Because MySQL supports character set conversion, it is important to separate IANA `shift_jis` and `cp932` into two different character sets because they provide different conversion rules.

How does cp932 differ from sjis?

The cp932 character set differs from `sjis` in the following ways:

- cp932 supports NEC special characters, NEC selected — IBM extended characters, and IBM selected characters.
- Some cp932 characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to cp932, one of the code points must be selected. For this “round trip conversion,” the rule recommended by Microsoft is used. (See <http://support.microsoft.com/kb/170559/EN-US/>.)

The conversion rule works like this:

- If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
- If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.
- If the character is in both IBM selected characters and NEC selected — IBM extended characters, use the code point of IBM extended characters.

The table shown at

<http://www.microsoft.com/globaldev/reference/dbcs/932.htm> provides information about the Unicode values of cp932 characters. For cp932 table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (`ucs2`) encoding. For table entries with an underlined two-digit value appears, there is a range of cp932

character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the cp932 characters that begin with those digits.

The following links are of special interest. They correspond to the encodings for the following sets of characters:

- NEC special characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_8

- NEC selected — IBM extended characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_EI
http://www.microsoft.com/globaldev/reference/dbcs/932/932_EI

- IBM selected characters:

http://www.microsoft.com/globaldev/reference/dbcs/932/932_F/
http://www.microsoft.com/globaldev/reference/dbcs/932/932_F/
http://www.microsoft.com/globaldev/reference/dbcs/932/932_F/

- Starting from version 5.0.3, cp932 supports conversion of user-defined characters in combination with eucjpm, and solves the problems with sjis/ujis conversion. For details, please refer to <http://www.opengroup.or.jp/jvc/cde/sjis-euc-e.html>.

For some characters, conversion to and from ucs2 is different for sjis and cp932. The following tables illustrate these differences.

Conversion to ucs2:

sjis/cp932 Value	sjis -> ucs2 Conversion	cp932 -> ucs2 Conversion
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225

817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

Conversion from ucs2:

ucs2 value	ucs2 -> sjis Conversion	ucs2 -> cp932 Conversion
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

Users of any Japanese character sets should be aware that using `--character-set-client-handshake` (or `--skip-character-set-client-handshake`) has an important effect. See [Section 5.2.1, “mysql Command Options”](#).

10.10. FAQ: MySQL Chinese, Japanese, and Korean Character Sets

This Frequently-Asked-Questions section comes from the experiences of MySQL's Support and Development groups, after handling many enquiries about CJK (Chinese Japanese Korean) issues.

10.10.1. SELECT shows non-Latin characters as "?"s. Why?

You inserted CJK characters with `INSERT`, but when you do a `SELECT`, they all look like "?". It usually is a setting in MySQL that doesn't match the settings for the application program or the operating system. These are common troubleshooting steps:

- Find out: what version do you have? The statement `SELECT VERSION();` will tell you. This FAQ is for MySQL version 5, so some of the answers here will not apply to you if you have version 4.0 or 4.1.
- Find out: what character set is the database column really in? Too frequently, people think that the character set will be the same as the server's set (false), or the set used for display purposes (false). Make sure, by saying `SHOW CREATE TABLE tablename,` or better yet by saying this:

```
SELECT character_set_name, collation_name
FROM   information_schema.columns WHERE table_schema = your_dat
AND    table_name = your_table_name
AND    column_name = your_column_name;
```

- Find out: what is the hexadecimal value?

```
SELECT HEX(your_column_name)
FROM your_table_name;
```

If you see `3F`, then that really is the encoding for `?`, so no wonder you see "?". Probably this happened because of a problem converting a particular character from your client character set to the target character set.

- Find out: is a literal round trip possible, that is, if you select "literal" (or "_introducer hexadecimal-value") do you get "literal" as a result? For

example, with the Japanese Katakana Letter Pe, which looks like 'ペ', and which exists in all CJK character sets, and which has the code point value (hexadecimal coding) 0x30da, enter:

```
SELECT 'ペ' AS `ペ`;          /* or SELECT _ucs2 0x30da; */
```

If the result doesn't look like ペ, a round trip failed. For bug reports, we might ask people to follow up with `SELECT hex('ペ');`. Then we can see whether the client encoding is right.

- Find out: is it the browser or application? Just use **mysql** (the MySQL client program, which on Windows will be **mysql.exe**). If **mysql** displays correctly but your application doesn't, then your problem is probably “Settings”, but consult also the question about “Troubles with Access (or Perl) (or PHP) (etc.)” much later in this FAQ.

To find your settings, the statement you need here is `SHOW VARIABLES`. For example:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name          | Value          |
+-----+-----+
| character_set_client   | utf8           |
| character_set_connection | utf8           |
| character_set_database | latin1         |
| character_set_filesystem | binary        |
| character_set_results  | utf8           |
| character_set_server   | latin1         |
| character_set_system   | utf8           |
| character_sets_dir     | /usr/local/mysql/share/mysql/charsets/
+-----+-----+
8 rows in set (0.03 sec)
```

The above are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set and a default for MySQL).

Although Unicode (usually the `utf8` variant on Unix, usually the `ucs2` variant on Windows) is better than “latin”, it's often not what your operating system utilities support best. Many Windows users find that a Microsoft character set, such as `cp932` for Japanese Windows, is what's suitable.

If you can't control the server settings, and you have no idea what your underlying computer is about, then try changing to a common character set for the country that you're in (euckr = Korea, gb2312 or gbk = People's Republic of China, big5 = other China, sjis or ujis or cp932 or eucjpm = Japan, ucs2 or utf8 = anywhere). Usually it is only necessary to change the client and connection and results settings, and there is a simple statement which changes all three at once, namely SET NAMES. For example:

```
SET NAMES 'big5';
```

Once you get the correct setting, you can make it permanent by editing my.cnf or my.ini. For example you might add lines looking like this:

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

10.10.2. Troubles with GB character sets (Chinese)

MySQL supports the two common variants of the GB (“Guojia Biaozhun” or “National Standard”) character sets which are official in the People's Republic of China: gb2312 and gbk. Sometimes people try to insert gbk characters into gb2312, and it works most of the time because gbk is a superset of gb2312. But eventually they try to insert a rarer Chinese character and it doesn't work. (Example: bug #16072 in our bugs database, <http://bugs.mysql.com/bug.php?id=16072>). So we'll try to clarify here exactly what characters are legitimate in gb2312 or gbk, with reference to the official documents. Please check these references before reporting gb2312 or gbk bugs. We now have a graphic listing of the gbk characters, currently on the site of Mr Alexander Barkov (MySQL's principal programmer for character set issues). The chart is in order according to the gb2312_chinese_ci collation:

http://d.udm.net/bar/~bar/charts/gb2312_chinese_ci.html. MySQL's gbk is in reality “Microsoft code page 936”. This differs from the official gbk for characters A1A4 (middle dot), A1AA (em dash), A6E0-A6F5, and A8BB-A8C0. For a listing of the differences, see <http://recode.progiciels-bpi.ca/showfile.html?name=dist/libiconv/gbk.h>. For a listing of gbk/Unicode mappings, see <http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/C>. For MySQL's listing of gbk characters, see http://d.udm.net/bar/~bar/charts/gbk_chinese_ci.html.

10.10.3. Troubles with big5 character set (Chinese)

MySQL supports the Big5 character set which is common in Hong Kong and the Republic of China (Taiwan). MySQL's big5 is in reality “Microsoft code page 950”, which is very similar to the original big5 character set. This is a recent change, starting with MySQL version 4.1.16 / 5.0.16. We made the change as a result of a bug report, bug #12476 in our bugs database, <http://bugs.mysql.com/bug.php?id=12476> (title: “Some big5 codes are still missing ...”). For example, the following statements work in the current version of MySQL, but not in old versions:

```
mysql> create table big5 (big5 char(1) character set big5);
Query OK, 0 rows affected (0.13 sec)
```

```
mysql> insert into big5 values (0xf9dc);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from big5;
```

```
+-----+
| big5 |
+-----+
|     |
+-----+
```

```
1 row in set (0.02 sec)
```

There is a feature request for adding HKSCS extensions (bug #13577 in our bugs database, <http://bugs.mysql.com/bug.php?id=13577>). People who need the extension may find the suggested patch for bug #13577 is of interest.

10.10.4. Troubles with character-set conversions (Japanese)

MySQL supports the sjis, ujis, cp932, and eucjpm character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with sjis or ujis) and a Windows client (typically with cp932). But conversions can seem to fail. Here's why. In this conversion table, the ucs2 column is the source, and the sjis/cp932/ujis/eucjpm columns are the destination, that is, what the hexadecimal result would be if we used CONVERT(ucs2) or if we assigned a ucs2 column containing the value to an sjis/cp932/ujis/eucjpm column.

character name	ucs2	sjis	cp932	ujis	eucjpm
-----	----	----	----	----	-----

BROKEN BAR	00A6	3F	3F	8FA2C3	3F
FULLWIDTH BROKEN BAR	FFE4	3F	FA55	3F	8FA2
YEN SIGN	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN	FFE5	818F	818F	A1EF	3F
TILDE	007E	7E	7E	7E	7E
OVERLINE	203E	3F	3F	20	3F
HORIZONTAL BAR	2015	815C	815C	A1BD	A1BD
EM DASH	2014	3F	3F	3F	3F
REVERSE SOLIDUS	005C	815F	5C	5C	5C
FULLWIDTH ""	FF3C	3F	815F	3F	A1C0
WAVE DASH	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE	2016	8161	3F	A1C2	3F
PARALLEL TO	2225	3F	8161	3F	A1C2
MINUS SIGN	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS	FF0D	3F	817C	3F	A1DD
CENT SIGN	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN	FFE0	3F	8191	3F	A1F1
POUND SIGN	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN	FFE1	3F	8192	3F	A1F2
NOT SIGN	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA	3F	A2CC

For example, consider this extract from the table:

	ucs2	sjis	cp932
	----	----	-----
NOT SIGN	00AC	81CA	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA

It means “for NOT SIGN which is Unicode U+00AC, MySQL converts to sjis code point 0x81CA and to cp932 code point 3F”. (3F is question mark (“?”) and is what we always use when we can't convert.) Now, what should we do if we want to convert sjis 81CA to cp932? Our answer is: “?”. There are serious complaints about this, many people would prefer a “loose” conversion, so that 81CA (NOT SIGN) in sjis becomes 81CA (FULLWIDTH NOT SIGN) in cp932. We

are considering changing.

10.10.5. The Great Yen Sign problem (Japanese)

In SJIS the code for Yen Sign (¥) is 5c. In SJIS the code for Reverse Solidus (\) is 5c. Since the above statements are contradictory, confusion often results. Well, to put it more seriously, some versions of Japanese character sets (both sjis and euc) have treated 5c as a reverse solidus, also known as a backslash, and others have treated it as a yen sign. There's nothing we can do, except take sides: MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description, and *5C is Reverse Solidus*, always. Should we make a separate character set where 5c is Yen Sign, as another DBMS (Oracle) does? We haven't decided. Certainly not in version 5.1 or 5.2. But if people keep complaining about The Great Yen Sign Problem, that's one possible solution.

10.10.6. Troubles with euckr character set (Korean)

MySQL supports the euckr (Extended Unix Code Korea) character set which is common in South Korea. In theory, problems could arise because there have been several versions of this character set. So far, only one problem has been noted, for Korea's currency symbol. We use the “ASCII” variant of EUC-KR, in which the code point 0x5c is REVERSE SOLIDUS, that is \, instead of the “KS-Roman” variant of EUC-KR, in which the code point 0x5c is WON SIGN, that is “₩”. You can't convert Unicode U+20A9 WON SIGN to euckr:

```
mysql> SELECT CONVERT('₩' USING euckr) AS euckr,  
-> HEX(CONVERT('₩' USING euckr)) AS hexeuckr;  
+-----+-----+  
| euckr | hexeuckr |  
+-----+-----+  
| ?     | 3F       |  
+-----+-----+  
1 row in set (0.00 sec)
```

MySQL's graphic Korean chart is here:

http://d.udm.net/bar/~bar/charts/euckr_korean_ci.html.

10.10.7. The “Data truncated” message

For illustration, we'll make a table with one Unicode (ucs2) column and one

Chinese (gb2312) column.

```
mysql> CREATE TABLE ch
-> (ucs2 CHAR(3) CHARACTER SET ucs2,
-> gb2312 CHAR(3) CHARACTER SET gb2312);
Query OK, 0 rows affected (0.05 sec)
```

We'll try to place the rare character in both columns.

```
mysql> INSERT INTO ch VALUES ('AB', 'AB');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

Ah, there's a warning. Let's see what it is.

```
mysql> SHOW WARNINGS;
+-----+-----+-----+-----+
| Level   | Code | Message                                     |
+-----+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'gb2312' at row 1 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

So it's a warning about the gb2312 column only.

```
mysql> SELECT ucs2, HEX(ucs2), gb2312, HEX(gb2312) FROM ch;
+-----+-----+-----+-----+
| ucs2   | HEX(ucs2)   | gb2312 | HEX(gb2312) |
+-----+-----+-----+-----+
| AB    | 00416C4C0042 | A?B    | 413F42      |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

There are several things that need explanation here.

1. The fact that it's a “warning” rather than an “error” is characteristic of MySQL. We like to try to do what we can, to get the best fit, rather than give up.
2. The character isn't in the gb2312 character set. We described that problem earlier.
3. Admittedly the message is misleading. We didn't “truncate” in this case, we replaced with a question mark. We've had a complaint about this message (bug #9337). But until we come up with something better, just accept that

error/warning code 2165 can mean a variety of things.

4. With SQL_MODE=TRADITIONAL, there would be an error message, but instead of error 2165 you would see: ERROR 1406 (22001): Data too long for column 'gb2312' at row 1.

10.10.8. Troubles with Access, Perl, PHP, etc.

You can't get things to look right with your special program for a GUI front end or browser? Get a direct connection to the server (with **mysql** on Unix or with **mysql.exe** on Windows) and try the same query there. If mysql is okay, then the trouble is probably that your application interface needs some initializing. Use **mysql** to tell you what character set(s) it uses, by saying `SHOW VARIABLES LIKE 'char%';`. If it's Access, you're probably connecting with MyODBC. So you'll want to check out the Reference Manual page for configuring an ODBC DSN, and pay attention particularly to the illustrations for “SQL command on connect”. You should enter `SET NAMES 'big5'` (supposing that you use big5) (you don't need a ; here). If it's ASP, you might need to add `SET NAMES` in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=yourserver;uid
    & "pwd=yourpassword;database=yourdatabase;stmt=SET NAMES 'bi
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

If it's PHP, here's a slightly different user suggestion:

```
<?php
$link = mysql_connect($host,$usr,$pwd);
mysql_select_db($db);
if (mysql_error()) { print "Database ERROR: " . mysql_error(); }
mysql_query("SET CHARACTER SET utf8", $link);
mysql_query("SET NAMES 'utf8'", $link);
?>
```

In this case, the tipper used `SET CHARACTER SET` statement to change `character_set_client` and `character_set_result`, and used `SET NAMES` to change `character_set_client` and `character_set_connection` and

character_set_results. So actually the SET CHARACTER SET statement is redundant. (Incidentally, MySQL people encourage the use of the `mysql` extension, rather than the `mysql` example that this example uses.) Another thing to check with PHP is the browser assumptions. Sometimes a meta tag change in the heading area suffices, for example: `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">`

For Connector/J tips, see the manual section in the Connectors chapter titled “Using Character Sets and Unicode”.

10.10.9. How can I get old MySQL 4.0 behaviour back?

In the old days, with MySQL Version 4.0, there was a single “global” character set for both server and client sides, and the decision was made by the server administrator. We changed that starting with MySQL Version 4.1. What happens now is a “handshake”. The MySQL Reference Manual describes it thus:

When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a SET NAMES operation using the character set name.

The effect of this is: you can't control the client character set by saying `mysqld --character-set-server=utf8`. But some Asian customers said that they don't like that, they want the MySQL 4.0 behaviour. So we added a **mysqld** switch, `--character-set-client-handshake`, which (and this is the interesting part) can be turned off with `--skip-character-set-client-handshake`. If you start `mysqld` with `--skip-character-set-client-handshake`, then the behaviour is like this: When a client connects, it sends to the server the name of the character set that it wants to use. The server ignores it! Here is an illustration with the handshake switch on or off. Pretend that your favourite server character set is `latin1` (of course that's unlikely in a CJK area but it's MySQL's default if there's no `my.ini` or `my.cnf` file). Pretend that the client operates with `utf8` because that's what the client's operating system supports. Start the server with a default character set, `latin1`:

```
mysqld --character-set-server=latin1
```

Start the client with a default character set, `utf8`:

```
mysql --default-character-set=utf8
```

Show what the current settings are:

```
mysql> SHOW VARIABLES LIKE 'char%';
```

```
+-----+-----+
| Variable_name          | Value
+-----+-----+
| character_set_client   | utf8
| character_set_connection | utf8
| character_set_database | latin1
| character_set_filesystem | binary
| character_set_results  | utf8
| character_set_server   | latin1
| character_set_system   | utf8
| character_sets_dir     | /usr/local/mysql/share/mysql/charsets/
+-----+-----+
8 rows in set (0.01 sec)
```

Stop the client. Stop the server with **mysqldadmin**. Start the server again but this time say “skip the handshake”:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handsh
```

Start the client with a default character set, utf8, again. Show what the current settings are, again:

```
mysql> SHOW VARIABLES LIKE 'char%';
```

```
+-----+-----+
| Variable_name          | Value
+-----+-----+
| character_set_client   | latin1
| character_set_connection | latin1
| character_set_database | latin1
| character_set_filesystem | binary
| character_set_results  | latin1
| character_set_server   | latin1
| character_set_system   | utf8
| character_sets_dir     | /usr/local/mysql/share/mysql/charsets/
+-----+-----+
8 rows in set (0.01 sec)
```

As you can see by comparing the SHOW VARIABLES results, the server ignores the client's initial settings if the --skip-character-set-client-handshake is used.

10.10.10. Why do some LIKE and FULLTEXT searches fail?

There is a simple problem with LIKE searches on BINARY and BLOB columns: we need to know the end of a character. With multi-byte character sets, different characters might have different octet lengths. For example, in utf8, A requires one byte but requires three bytes. Illustration:

```

+-----+-----+
| octet_length(_utf8 'A') | octet_length(_utf8 '') |
+-----+-----+
|                1 |                3 |
+-----+-----+
1 row in set (0.00 sec)

```

If we don't know where the first character ends, then we don't know where the second character begins, and even simple-looking searches like LIKE '_A%' will fail. The solution is to use a regular CJK character set in the first place, or convert to a CJK character character set before comparing. Incidentally, this is one reason why MySQL cannot allow encodings of nonexistent characters: It must be strict about rejecting bad input, or it won't know where characters end. There is a simple problem with FULLTEXT: we need to know the end of a word. With Western writing this is rarely a problem because there are spaces between words. With Asian writing this is not the case. We could use half-good solutions, like saying that all Han characters represent words, or depending on (Japanese) changes from Katakana to Hiragana which are due to grammatical endings. But the only good solution requires a dictionary, and we haven't found a good open-source dictionary.

10.10.11. What CJK character sets are available?

The list of CJK character sets may vary depending on version. For example, the eucjpm character set is a recent addition. But the language name appears in the DESCRIPTION column for every entry in information_schema.character_sets. Therefore, to get a current list of all the non-Unicode CJK character sets, say:

```

mysql> SELECT character_set_name, description
-> FROM information_schema.character_sets
-> WHERE description LIKE '%Chinese%'
-> OR description LIKE '%Japanese%'
-> OR description LIKE '%Korean%'
-> ORDER BY character_set_name;
+-----+-----+
| character_set_name | description |

```

```

+-----+-----+
| big5      | Big5 Traditional Chinese |
| cp932     | SJIS for Windows Japanese |
| eucjpms   | UJIS for Windows Japanese |
| euckr     | EUC-KR Korean            |
| gb2312    | GB2312 Simplified Chinese |
| gbk       | GBK Simplified Chinese    |
| sjis      | Shift-JIS Japanese       |
| ujis      | EUC-JP Japanese          |
+-----+-----+
8 rows in set (0.01 sec)

```

10.10.12. Is character X available in all character sets?

The majority of everyday-use Chinese/Japanese characters (simplified Chinese and basic non-halfwidth Kana Japanese) appear in all CJK character sets. Here is a stored procedure which accepts a UCS-2 Unicode character, converts it to all other character sets, and displays the results in hexadecimal.

```
DELIMITER //
```

```
CREATE PROCEDURE p_convert (ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN
```

```
CREATE TABLE tj
(ucs2 CHAR(1) character set ucs2,
 utf8 CHAR(1) character set utf8,
 big5 CHAR(1) character set big5,
 cp932 CHAR(1) character set cp932,
 eucjpms CHAR(1) character set eucjpms,
 euckr CHAR(1) character set euckr,
 gb2312 CHAR(1) character set gb2312,
 gbk CHAR(1) character set gbk,
 sjis CHAR(1) character set sjis,
 ujis CHAR(1) character set ujis);
```

```
INSERT INTO tj (ucs2) VALUES (ucs2_char);
```

```
UPDATE tj SET utf8=ucs2,
 big5=ucs2,
 cp932=ucs2,
 eucjpms=ucs2,
 euckr=ucs2,
 gb2312=ucs2,
 gbk=ucs2,
 sjis=ucs2,
 ujis=ucs2;
```

```

/* If there's a conversion problem, UPDATE will produce a warning. */
SELECT hex(ucs2) AS ucs2,
       hex(utf8) AS utf8,
       hex(big5) AS big5,
       hex(cp932) AS cp932,
       hex(eucjpms) AS eucjpms,
       hex(euckr) AS euckr,
       hex(gb2312) AS gb2312,
       hex(gbk) AS gbk,
       hex(sjis) AS sjis,
       hex(ujis) AS ujis
FROM tj;

DROP TABLE tj;

END//

```

The input can be any single ucs2 character, or it can be the code point value (hexadecimal representation) of that character. Here's an example of what P_CONVERT() can do. An earlier answer said that the character “Katakana Letter Pe” appears in all CJK character sets. We know that the code point value of Katakana Letter Pe is 0x30da. (By the way, we got the name from Unicode's list of ucs2 encodings and names:

<http://www.unicode.org/Public/UNIDATA/UnicodeData.txt>.) So we'll say:

```

mysql> CALL P_CONVERT(0x30da)//
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ucs2 | utf8   | big5  | cp932 | eucjpms | euckr | gb2312 | gbk   | s
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 30DA | E3839A | C772  | 8379  | A5DA    | ABDA  | A5DA   | A5DA  | 8
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)

```

Since none of the column values is 3F, we know that every conversion worked.

10.10.13. Strings don't sort correctly in Unicode (I)

Sometimes people observe that the result of a utf8_unicode_ci or ucs2_unicode_ci search or ORDER BY sort is not what they think a native would expect. Although we never rule out the chance that there is a bug, we have found in the past that people are not correctly reading the standard table of weights for the Unicode Collation Algorithm. So, here's how to check whether we're using

the right collation. The correct table for MySQL is this one: <http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt>. This is different from the first table you will find by navigating from the `unicode.org` home page. MySQL deliberately uses the older 4.0.0 “allkeys” table, instead of the current 4.1.0 table. We are very wary about changing ordering which affects indexes. Here is an example of a problem that we handled recently, for a complaint in our bugs database, <http://bugs.mysql.com/bug.php?id=16526>:

```
mysql> CREATE TABLE tj (s1 CHAR(1) CHARACTER SET utf8 COLLATE utf8_u
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> INSERT INTO tj VALUES (''), ('');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM tj WHERE s1 = '';
```

```
+-----+
| s1    |
+-----+
|      |
|      |
+-----+
2 rows in set (0.00 sec)
```

If your eyes are sharp, you'll see that the character in the first result row isn't the one that we searched for. Why did MySQL retrieve it? First we look for the Unicode code point value, which is possible by reading the hexadecimal number for the `ucs2` version of the characters:

```
mysql> SELECT s1, HEX(CONVERT(s1 USING ucs2)) FROM tj;
```

```
+-----+-----+
| s1    | HEX(CONVERT(s1 USING ucs2)) |
+-----+-----+
|      | 304C                          |
|      | 304B                          |
+-----+-----+
2 rows in set (0.03 sec)
```

Now let's search for 304B and 304C in the 4.0.0 allkeys table. We'll find these lines:

```
304B ; [.1E57.0020.000E.304B] # HIRAGANA LETTER KA
304C ; [.1E57.0020.000E.304B][.0000.0140.0002.3099] # HIRAGANA LETT
```

The official Unicode names (following the “#” mark) are informative; they tell

us the Japanese syllabary (Hiragana), the informal classification (letter instead of digit or punctuation), and the Western identifier (KA or GA, which happen to be voiced/unvoiced components of the same letter pair). More importantly, the Primary Weight (the first hexadecimal number inside the square brackets) is 1E57 on both lines. For comparisons in both searching and sorting, MySQL pays attention only to the Primary Weight, it ignores all the other numbers. So now we know that we're sorting and correctly according to the Unicode specification. If we wanted to distinguish them, we'd have to use a non-Unicode-Collation-Algorithm collation (`utf8_unicode_bin` or `utf8_general_ci`), or compare the `HEX()` values, or say `ORDER BY CONVERT(s1 USING sjis)`. Being correct "according to Unicode" isn't enough, of course: the person who submitted the bug was equally correct. We plan to add another collation for Japanese according to the JIS X 4061 standard, where voiced/unvoiced letters like KA/GA are distinguishable for ordering purposes.

10.10.14. Strings don't sort correctly in Unicode (II)

You're using Unicode (`ucs2` or `utf8`), and you know what the Unicode sort order is (see the previous question and answer), but MySQL still seems to sort your table wrong? This might be easy.

```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
`s1` char(1) CHARACTER SET ucs2 DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Hmm, the character set looks okay. Let's look at the `information_schema` for this column.

```
mysql> SELECT column_name, character_set_name, collation_name
-> FROM information_schema.columns
-> WHERE column_name = 's1'
-> AND table_name = 't';
+-----+-----+-----+
| column_name | character_set_name | collation_name |
+-----+-----+-----+
| s1          | ucs2                | ucs2_general_ci |
+-----+-----+-----+
1 row in set (0.01 sec)
```

Oops, the collation is `ucs2_general_ci` instead of `ucs2_unicode_ci`! Here's why:

```
mysql> SHOW CHARSET LIKE 'ucs2%';
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| ucs2    | UCS-2 Unicode | ucs2_general_ci  |      2 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

For `ucs2` and `utf8`, the “general” collation is the default. To specify that you wanted a “unicode” collation, you should have specified `COLLATE ucs2_unicode_ci`.

10.10.15. My supplementary characters get rejected

Right. MySQL doesn't support supplementary characters (characters which need more than 3 bytes with UTF-8). We support only what Unicode calls the *Basic Multilingual Plane / Plane 0*. Only a few very rare Han characters are supplementary; support for them is uncommon. This has led to bug #12600 (<http://bugs.mysql.com/bug.php?id=12600>) which we rejected as “not a bug”. With `utf8`, we must truncate an input string when we encounter bytes that we don't understand. Otherwise, we wouldn't know how long the bad multi-byte character is. A workaround is: if you use `ucs2` instead of `utf8`, then the bad characters will change to question marks, but there will be no truncation. Or change the data type to `BLOB` or `BINARY`, which have no validity checking. In our bugs database, bug #14052 (<http://bugs.mysql.com/bug.php?id=14052>) is a feature request for Wikipedia, asking us to support supplementary characters extending `ucs2` as well as `utf8`.

10.10.16. Shouldn't it be CJKV (V for Vietnamese)?

No. The term CJKV (Chinese Japanese Korean Vietnamese) refers to character sets which contain Han (originally Chinese) characters. MySQL has no plan to support the old Vietnamese script using Han characters. MySQL does of course support the modern Vietnamese script with Western characters. Another question that has come up (once) is a request for specialized Vietnamese collation, see <http://bugs.mysql.com/bug.php?id=4745>. We might do something about it someday, if many more requests arise.

10.10.17. Will MySQL fix any CJK problems in version 5.1?

Yes. We're changing the names of files and directories. Here's an example, using mysql as root under Linux:

1. Create a table with a name containing a Han character:

```
mysql> CREATE TABLE tab_ (s1 INT);
Query OK, 0 rows affected (0.07 sec)
```

2. Find out where MySQL stores database files:

```
mysql> SHOW VARIABLES LIKE 'datadir';
+-----+-----+
| Variable_name | Value                |
+-----+-----+
| datadir       | /usr/local/mysql/var/ |
+-----+-----+
1 row in set (0.00 sec)
```

3. Look at the directory to see the MyISAM table files:

```
# cd /usr/local/mysql/var/dba
# dir tab_*
-rw-rw----  1 root root    0 2006-05-16 10:22 tab_@696e.MYD
-rw-rw----  1 root root 1024 2006-05-16 10:22 tab_@696e.MYI
-rw-rw----  1 root root 8556 2006-05-16 10:22 tab_@696e.frm
```

Notice that MySQL has converted the Han character to @ + (Unicode value of Han character), that is, to a purely ASCII representation. This solves an old problem, that database files weren't portable, because some computers wouldn't allow in a file name. Conversion to the new file names will be automatic when you upgrade to version 5.1. This should take care of bug #6313 in our bugs database, <http://bugs.mysql.com/bug.php?id=6313>.

10.10.18. When will MySQL translate the manual again?

A Beijing-based group has produced a Simplified Chinese version for us under contract. It's complete and can be found on <http://dev.mysql.com/doc/#chinese-5.1>. It's up to date as of version 5.1.2. The Japanese manual can be downloaded from <http://dev.mysql.com/doc/#japanese-4.1>. It is still for version 4.1.

10.10.19. Whom can I talk to?

Check <http://dev.mysql.com/user-groups/> to see if there is a MySQL user group near you. If there isn't: why not start one yourself? To contact a sales engineer in MySQL KK's Japan office:

Tel: +81(0)3-5326-3133
Fax: +81(0)3-5326-3001
Email: dsaito@mysql.com

To see feature requests about language issues:

- Go to <http://bugs.mysql.com>.
- Click Advanced Search.
- In the Severity dropdown box, click S4 (Feature Request).
- In the list box beside Category, click Character Sets.
- Click the Search button.

You can post CJK questions, or see previous answers, on MySQL's “Character Sets, Collation, Unicode” forum: <http://forums.mysql.com/list.php?103>. MySQL plans to add native-language forums on <http://forums.mysql.com/> very soon.

Chapter 11. Data Types

Table of Contents

[11.1. Data Type Overview](#)

[11.1.1. Overview of Numeric Types](#)

[11.1.2. Overview of Date and Time Types](#)

[11.1.3. Overview of String Types](#)

[11.1.4. Data Type Default Values](#)

[11.2. Numeric Types](#)

[11.3. Date and Time Types](#)

[11.3.1. The DATETIME, DATE, and TIMESTAMP Types](#)

[11.3.2. The TIME Type](#)

[11.3.3. The YEAR Type](#)

[11.3.4. Y2K Issues and Date Types](#)

[11.4. String Types](#)

[11.4.1. The CHAR and VARCHAR Types](#)

[11.4.2. The BINARY and VARBINARY Types](#)

[11.4.3. The BLOB and TEXT Types](#)

[11.4.4. The ENUM Type](#)

[11.4.5. The SET Type](#)

[11.5. Data Type Storage Requirements](#)

[11.6. Choosing the Right Type for a Column](#)

[11.7. Using Data Types from Other Database Engines](#)

MySQL supports a number of data types in several categories: numeric types, date and time types, and string (character) types. This chapter first gives an overview of these data types, and then provides a more detailed description of the properties of the types in each category, and a summary of the data type storage requirements. The initial overview is intentionally brief. The more detailed descriptions later in the chapter should be consulted for additional information about particular data types, such as the allowable formats in which you can specify values.

MySQL also supports extensions for handling spatial data. [Chapter 16, Spatial Extensions](#), provides information about these data types.

Several of the data type descriptions use these conventions:

- M indicates the maximum display width for integer types. For floating-point and fixed-point types, M is the total number of digits. For string types, M is the maximum length. The maximum allowable value of M depends on the data type.
- D applies to floating-point and fixed-point types and indicates the number of digits following the decimal point. The maximum possible value is 30, but should be no greater than $M-2$.
- Square brackets (‘[’ and ‘]’) indicate optional parts of type definitions.

11.1. Data Type Overview

11.1.1. Overview of Numeric Types

A summary of the numeric data types follows. For additional information, see [Section 11.2, “Numeric Types”](#). Storage requirements are given in [Section 11.5, “Data Type Storage Requirements”](#).

M indicates the maximum display width. The maximum legal display width is 255. Display width is unrelated to the storage size or range of values a type can contain, as described in [Section 11.2, “Numeric Types”](#).

If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute to the column.

SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.

SERIAL DEFAULT VALUE in the definition of an integer column is an alias for NOT NULL AUTO_INCREMENT UNIQUE.

Warning: When you use subtraction between integer values where one is of type UNSIGNED, the result is unsigned unless the NO_UNSIGNED_SUBTRACTION SQL mode is enabled. See [Section 12.8, “Cast Functions and Operators”](#).

- BIT[(M)]

A bit-field type. *M* indicates the number of bits per value, from 1 to 64. The default is 1 if *M* is omitted.

This data type was added in MySQL 5.0.3 for MyISAM, and extended in 5.0.5 to MEMORY, InnoDB, and BDB. Before 5.0.3, BIT is a synonym for TINYINT(1).

- TINYINT[(M)] [UNSIGNED] [ZEROFILL]

A very small integer. The signed range is -128 to 127. The unsigned range is 0 to 255.

- **BOOL, BOOLEAN**

These types are synonyms for `TINYINT(1)`. A value of zero is considered false. Non-zero values are considered true:

```
mysql> SELECT IF(0, 'true', 'false');
+-----+
| IF(0, 'true', 'false') |
+-----+
| false                   |
+-----+
```

```
mysql> SELECT IF(1, 'true', 'false');
+-----+
| IF(1, 'true', 'false') |
+-----+
| true                   |
+-----+
```

```
mysql> SELECT IF(2, 'true', 'false');
+-----+
| IF(2, 'true', 'false') |
+-----+
| true                   |
+-----+
```

However, the values `TRUE` and `FALSE` are merely aliases for 1 and 0, respectively, as shown here:

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+-----+
| IF(0 = FALSE, 'true', 'false') |
+-----+
| true                           |
+-----+
```

```
mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-----+
| IF(1 = TRUE, 'true', 'false') |
+-----+
| true                           |
+-----+
```

```
mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-----+
| IF(2 = TRUE, 'true', 'false') |
+-----+
| false                           |
+-----+
```

```

+-----+
mysql> SELECT IF(2 = FALSE, 'true', 'false');
+-----+
| IF(2 = FALSE, 'true', 'false') |
+-----+
| false                            |
+-----+

```

The last two statements display the results shown because 2 is equal to neither 1 nor 0.

We intend to implement full boolean type handling, in accordance with standard SQL, in a future MySQL release.

- **SMALLINT[(M)] [UNSIGNED] [ZEROFILL]**

A small integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.

- **MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]**

A medium-sized integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215.

- **INT[(M)] [UNSIGNED] [ZEROFILL]**

A normal-size integer. The signed range is -2147483648 to 2147483647. The unsigned range is 0 to 4294967295.

- **INTEGER[(M)] [UNSIGNED] [ZEROFILL]**

This type is a synonym for INT.

- **BIGINT[(M)] [UNSIGNED] [ZEROFILL]**

A large integer. The signed range is -9223372036854775808 to 9223372036854775807. The unsigned range is 0 to 18446744073709551615.

Some things you should be aware of with respect to BIGINT columns:

- All arithmetic is done using signed BIGINT or DOUBLE values, so you

should not use unsigned big integers larger than 9223372036854775807 (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a `BIGINT` value to a `DOUBLE`.

MySQL can handle `BIGINT` in the following cases:

- When using integers to store large unsigned values in a `BIGINT` column.
 - In `MIN(col_name)` or `MAX(col_name)`, where *col_name* refers to a `BIGINT` column.
 - When using operators (+, -, *, and so on) where both operands are integers.
 - You can always store an exact integer value in a `BIGINT` column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
 - The -, +, and * operators use `BIGINT` arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than 9223372036854775807.
- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

M is the total number of decimal digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits allowed by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

`UNSIGNED`, if specified, disallows negative values.

Using `FLOAT` might give you some unexpected problems because all calculations in MySQL are done with double precision. See [Section A.5.7, “Solving Problems with No Matching Rows”](#).

- `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`

A normal-size (double-precision) floating-point number. Allowable values are `-1.7976931348623157E+308` to `-2.2250738585072014E-308`, `0`, and `2.2250738585072014E-308` to `1.7976931348623157E+308`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

M is the total number of decimal digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits allowed by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

`UNSIGNED`, if specified, disallows negative values.

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`, `REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

These types are synonyms for `DOUBLE`. Exception: If the `REAL_AS_FLOAT` SQL mode is enabled, `REAL` is a synonym for `FLOAT` rather than `DOUBLE`.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

A floating-point number. *p* represents the precision in bits, but MySQL uses this value only to determine whether to use `FLOAT` or `DOUBLE` for the resulting data type. If *p* is from 0 to 24, the data type becomes `FLOAT` with no *M* or *D* values. If *p* is from 25 to 53, the data type becomes `DOUBLE` with no *M* or *D* values. The range of the resulting column is the same as for the single-precision `FLOAT` or double-precision `DOUBLE` data types described earlier in this section.

`FLOAT(p)` syntax is provided for ODBC compatibility.

- `DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]`

For MySQL 5.0.3 and above:

A packed “exact” fixed-point number. M is the total number of decimal digits (the precision) and D is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the ‘-’ sign are not counted in M . If D is 0, values have no decimal point or fractional part. The maximum number of digits (M) for DECIMAL is 65 (64 from 5.0.3 to 5.0.5). The maximum number of supported decimals (D) is 30. If D is omitted, the default is 0. If M is omitted, the default is 10.

UNSIGNED, if specified, disallows negative values.

All basic calculations (+, -, *, /) with DECIMAL columns are done with a precision of 65 digits.

Before MySQL 5.0.3:

An unpacked fixed-point number. Behaves like a CHAR column; “unpacked” means the number is stored as a string, using one character for each digit of the value. M is the total number of digits and D is the number of digits after the decimal point. The decimal point and (for negative numbers) the ‘-’ sign are not counted in M , although space for them is reserved. If D is 0, values have no decimal point or fractional part. The maximum range of DECIMAL values is the same as for DOUBLE, but the actual range for a given DECIMAL column may be constrained by the choice of M and D . If D is omitted, the default is 0. If M is omitted, the default is 10.

UNSIGNED, if specified, disallows negative values.

The behavior used by the server for DECIMAL columns in a table depends on the version of MySQL used to create the table. If your server is from MySQL 5.0.3 or higher, but you have DECIMAL columns in tables that were created before 5.0.3, the old behavior still applies to those columns. To convert the tables to the newer DECIMAL format, dump them with **mysqldump** and reload them.

- DEC[(M[,D])] [UNSIGNED] [ZEROFILL], NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL], FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]

These types are synonyms for DECIMAL. The FIXED synonym is available for compatibility with other database systems.

11.1.2. Overview of Date and Time Types

A summary of the temporal data types follows. For additional information, see [Section 11.3, “Date and Time Types”](#). Storage requirements are given in [Section 11.5, “Data Type Storage Requirements”](#).

For the DATETIME and DATE range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The SUM() and AVG() aggregate functions do not work with temporal values. (They convert the values to numbers, which loses the part after the first non-numeric character.) To work around this problem, you can convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;  
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

- DATE

A date. The supported range is '1000-01-01' to '9999-12-31'. MySQL displays DATE values in 'YYYY-MM-DD' format, but allows you to assign values to DATE columns using either strings or numbers.

- DATETIME

A date and time combination. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. MySQL displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format, but allows you to assign values to DATETIME columns using either strings or numbers.

- TIMESTAMP

A timestamp. The range is '1970-01-01 00:00:00' to partway through the year 2037.

A TIMESTAMP column is useful for recording the date and time of an INSERT or UPDATE operation. By default, the first TIMESTAMP column in a table is automatically set to the date and time of the most recent operation if you do not assign it a value yourself. You can also set any TIMESTAMP column to the

current date and time by assigning it a NULL value. Variations on automatic initialization and update properties are described in [Section 11.3.1.1, “TIMESTAMP Properties as of MySQL 4.1”](#).

A `TIMESTAMP` value is returned as a string in the format 'YYYY-MM-DD HH:MM:SS' with a display width fixed at 19 characters. To obtain the value as a number, you should add `+0` to the timestamp column.

Note: The `TIMESTAMP` format that was used prior to MySQL 4.1 is not supported in MySQL 5.0; see *MySQL 3.23, 4.0, 4.1 Reference Manual* for information regarding the old format.

- `TIME`

A time. The range is '-838:59:59' to '838:59:59'. MySQL displays `TIME` values in 'HH:MM:SS' format, but allows you to assign values to `TIME` columns using either strings or numbers.

- `YEAR[(2|4)]`

A year in two-digit or four-digit format. The default is four-digit format. In four-digit format, the allowable values are 1901 to 2155, and 0000. In two-digit format, the allowable values are 70 to 69, representing years from 1970 to 2069. MySQL displays `YEAR` values in `YYYY` format, but allows you to assign values to `YEAR` columns using either strings or numbers.

11.1.3. Overview of String Types

A summary of the string data types follows. For additional information, see [Section 11.4, “String Types”](#). Storage requirements are given in [Section 11.5, “Data Type Storage Requirements”](#).

In some cases, MySQL may change a string column to a type different from that given in a `CREATE TABLE` or `ALTER TABLE` statement. See [Section 13.1.5.1, “Silent Column Specification Changes”](#).

In MySQL 4.1 and up, string data types include some features that you may not have encountered in working with versions of MySQL prior to 4.1:

- MySQL interprets length specifications in character column definitions in

character units. (Before MySQL 4.1, column lengths were interpreted in bytes.) This applies to CHAR, VARCHAR, and the TEXT types.

- Column definitions for many string data types can include attributes that specify the character set or collation of the column. These attributes apply to the CHAR, VARCHAR, the TEXT types, ENUM, and SET data types:

- The CHARACTER SET attribute specifies the character set, and the COLLATE attribute specifies a collation for the the character set. For example:

```
CREATE TABLE t
(
    c1 VARCHAR(20) CHARACTER SET utf8,
    c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
);
```

This table definition creates a column named c1 that has a character set of utf8 with the default collation for that character set, and a column named c2 that has a character set of latin1 and a case-sensitive collation.

CHARSET is a synonym for CHARACTER SET.

- The ASCII attribute is shorthand for CHARACTER SET latin1.
 - The UNICODE attribute is shorthand for CHARACTER SET ucs2.
 - The BINARY attribute is shorthand for specifying the binary collation of the column character set. In this case, sorting and comparison are based on numeric character values. (Before MySQL 4.1, BINARY caused a column to store binary strings and sorting and comparison were based on numeric byte values. This is the same as using character values for single-byte character sets, but not for multi-byte character sets.)
- Character column sorting and comparison are based on the character set assigned to the column. (Before MySQL 4.1, sorting and comparison were based on the collation of the server character set.) For the CHAR, VARCHAR, TEXT, ENUM, and SET data types, you can declare a column with a binary collation or the BINARY attribute to cause sorting and comparison to use the

underlying character code values rather than a lexical ordering.

[Chapter 10, Character Set Support](#), provides additional information about use of character sets in MySQL.

- `[NATIONAL] CHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

A fixed-length string that is always right-padded with spaces to the specified length when stored. *M* represents the column length. The range of *M* is 0 to 255 characters.

Note: Trailing spaces are removed when CHAR values are retrieved.

Before MySQL 5.0.3, a CHAR column with a length specification greater than 255 is converted to the smallest TEXT type that can hold values of the given length. For example, CHAR(500) is converted to TEXT, and CHAR(200000) is converted to MEDIUMTEXT. This is a compatibility feature. However, this conversion causes the column to become a variable-length column, and also affects trailing-space removal.

In MySQL 5.0.3 and later, if you attempt to set the length of a CHAR greater than 255, the CREATE TABLE or ALTER TABLE statement in which this is done fails with an error:

```
mysql> CREATE TABLE c1 (col1 INT, col2 CHAR(500));
ERROR 1074 (42000): Column length too big for column 'col' (max
use BLOB or TEXT instead
mysql> SHOW CREATE TABLE c1;
ERROR 1146 (42S02): Table 'test.c1' doesn't exist
```

CHAR is shorthand for CHARACTER. NATIONAL CHAR (or its equivalent short form, NCHAR) is the standard SQL way to define that a CHAR column should use some predefined character set. MySQL 4.1 and up uses utf8 as this predefined character set. [Section 10.3.6, “National Character Set”](#).

The CHAR BYTE data type is an alias for the BINARY data type. This is a compatibility feature.

MySQL allows you to create a column of type CHAR(0). This is useful primarily when you have to be compliant with old applications that depend

on the existence of a column but that do not actually use its value. CHAR(0) is also quite nice when you need a column that can take only two values: A column that is defined as CHAR(0) NULL occupies only one bit and can take only the values NULL and '' (the empty string).

- CHAR [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

This type is a synonym for CHAR(1).

- [NATIONAL] VARCHAR(M) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

A variable-length string. *M* represents the maximum column length. In MySQL 5.0, the range of *M* is 0 to 255 before MySQL 5.0.3, and 0 to 65,535 in MySQL 5.0.3 and later. (The actual maximum length of a VARCHAR in MySQL 5.0 is determined by the maximum row size and the character set you use. The maximum *effective* length starting with MySQL 5.0.3 is 65,532 bytes.)

Note: Before 5.0.3, trailing spaces were removed when VARCHAR values were stored, which differs from the standard SQL specification.

Prior to MySQL 5.0.3, a VARCHAR column with a length specification greater than 255 was converted to the smallest TEXT type that could hold values of the given length. For example, VARCHAR(500) was converted to TEXT, and VARCHAR(200000) was converted to MEDIUMTEXT. This was a compatibility feature. However, this conversion affected trailing-space removal.

VARCHAR is shorthand for CHARACTER VARYING.

VARCHAR values are stored using as many characters as are needed, plus one byte to record the length (two bytes for columns that are declared with a length longer than 255).

- BINARY(M)

The BINARY type is similar to the CHAR type, but stores binary byte strings rather than non-binary character strings.

- VARBINARY(M)

The VARBINARY type is similar to the VARCHAR type, but stores binary byte strings rather than non-binary character strings.

- TINYBLOB

A BLOB column with a maximum length of 255 ($2^8 - 1$) bytes.

- TINYTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

A TEXT column with a maximum length of 255 ($2^8 - 1$) characters.

- BLOB[(*M*)]

A BLOB column with a maximum length of 65,535 ($2^{16} - 1$) bytes.

An optional length *M* can be given for this type. If this is done, MySQL creates the column as the smallest BLOB type large enough to hold values *M* bytes long.

- TEXT[(*M*)] [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

A TEXT column with a maximum length of 65,535 ($2^{16} - 1$) characters.

An optional length *M* can be given for this type. If this is done, MySQL creates the column as the smallest TEXT type large enough to hold values *M* characters long.

- MEDIUMBLOB

A BLOB column with a maximum length of 16,777,215 ($2^{24} - 1$) bytes.

- MEDIUMTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

A TEXT column with a maximum length of 16,777,215 ($2^{24} - 1$) characters.

- LONGBLOB

A BLOB column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$)

bytes. The maximum *effective* (permitted) length of `LONGBLOB` columns depends on the configured maximum packet size in the client/server protocol and available memory.

- `LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

A `TEXT` column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) characters. The maximum *effective* (permitted) length of `LONGTEXT` columns depends on the configured maximum packet size in the client/server protocol and available memory.

- `ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]`

An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', . . . , `NULL` or the special ' ' error value. An `ENUM` column can have a maximum of 65,535 distinct values. `ENUM` values are represented internally as integers.

- `SET('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]`

A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', . . . A `SET` column can have a maximum of 64 members. `SET` values are represented internally as integers.

11.1.4. Data Type Default Values

The `DEFAULT value` clause in a data type specification indicates a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` or `CURRENT_DATE`. The exception is that you can specify `CURRENT_TIMESTAMP` as the default for a `TIMESTAMP` column. See [Section 11.3.1.1, “TIMESTAMP Properties as of MySQL 4.1”](#).

Prior to MySQL 5.0.2, if a column definition includes no explicit `DEFAULT` value, MySQL determines the default value as follows:

If the column can take NULL as a value, the column is defined with an explicit DEFAULT NULL clause.

If the column cannot take NULL as the value, MySQL defines the column with an explicit DEFAULT clause, using the implicit default value for the column data type. Implicit defaults are defined as follows:

- For numeric types other than integer types declared with the AUTO_INCREMENT attribute, the default is 0. For an AUTO_INCREMENT column, the default value is the next value in the sequence.
- For date and time types other than TIMESTAMP, the default is the appropriate “zero” value for the type. For the first TIMESTAMP column in a table, the default value is the current date and time. See [Section 11.3, “Date and Time Types”](#).
- For string types other than ENUM, the default value is the empty string. For ENUM, the default is the first enumeration value.

BLOB and TEXT columns cannot be assigned a default value.

As of MySQL 5.0.2, if a column definition includes no explicit DEFAULT value, MySQL determines the default value as follows:

If the column can take NULL as a value, the column is defined with an explicit DEFAULT NULL clause. This is the same as before 5.0.2.

If the column cannot take NULL as the value, MySQL defines the column with no explicit DEFAULT clause. For data entry, if an INSERT or REPLACE statement includes no value for the column, MySQL handles the column according to the SQL mode in effect at the time:

- If strict SQL mode is not enabled, MySQL sets the column to the implicit default value for the column data type.
- If strict mode is enabled, an error occurs for transactional tables and the statement is rolled back. For non-transactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows will have been inserted.

Suppose that a table `t` is defined as follows:

```
CREATE TABLE t (i INT NOT NULL);
```

In this case, `i` has no explicit default, so in strict mode each of the following statements produce an error and no row is inserted. When not using strict mode, only the third statement produces an error; the implicit default is inserted for the first two statements, but the third fails because `DEFAULT(i)` cannot produce a value:

```
INSERT INTO t VALUES();  
INSERT INTO t VALUES(DEFAULT);  
INSERT INTO t VALUES(DEFAULT(i));
```

See [Section 5.2.5, “The Server SQL Mode”](#).

For a given table, you can use the `SHOW CREATE TABLE` statement to see which columns have an explicit `DEFAULT` clause.

11.2. Numeric Types

MySQL supports all of the standard SQL numeric data types. These types include the exact numeric data types (INTEGER, SMALLINT, DECIMAL, and NUMERIC), as well as the approximate numeric data types (FLOAT, REAL, and DOUBLE PRECISION). The keyword INT is a synonym for INTEGER, and the keyword DEC is a synonym for DECIMAL. For numeric type storage requirements, see [Section 11.5, “Data Type Storage Requirements”](#).

As of MySQL 5.0.3, a BIT data type is available for storing bit-field values. (Before 5.0.3, MySQL interprets BIT as TINYINT(1).) In MySQL 5.0.3, BIT is supported only for MyISAM. MySQL 5.0.5 extends BIT support to MEMORY, InnoDB, and BDB.

As an extension to the SQL standard, MySQL also supports the integer types TINYINT, MEDIUMINT, and BIGINT. The following table shows the required storage and range for each of the integer types.

Type	Bytes	Minimum Value (Signed/Unsigned)	Maximum Value (Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

Another extension is supported by MySQL for optionally specifying the display width of an integer value in parentheses following the base keyword for the type (for example, INT(4)). This optional display width specification is used to left-

pad the display of values having a width less than the width specified for the column.

The display width does *not* constrain the range of values that can be stored in the column, nor the number of digits that are displayed for values having a width exceeding that specified for the column.

When used in conjunction with the optional extension attribute `ZEROFILL`, the default padding of spaces is replaced with zeros. For example, for a column declared as `INT(5) ZEROFILL`, a value of 4 is retrieved as `00004`. Note that if you store larger values than the display width in an integer column, you may experience problems when MySQL generates temporary tables for some complicated joins, because in these cases MySQL assumes that the data fits into the original column width.

Note: The `ZEROFILL` attribute is stripped when a column is involved in expressions or `UNION` queries.

All integer types can have an optional (non-standard) attribute `UNSIGNED`. Unsigned values can be used when you want to allow only non-negative numbers in a column and you need a larger upper numeric range for the column. For example, if an `INT` column is `UNSIGNED`, the size of the column's range is the same but its endpoints shift from `-2147483648` and `2147483647` up to `0` and `4294967295`.

Floating-point and fixed-point types also can be `UNSIGNED`. As with integer types, this attribute prevents negative values from being stored in the column. However, unlike the integer types, the upper range of column values remains the same.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

For floating-point data types, MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

The `FLOAT` and `DOUBLE` data types are used to represent approximate numeric data values. For `FLOAT` the SQL standard allows an optional specification of the precision (but not the range of the exponent) in bits following the keyword `FLOAT` in parentheses. MySQL also supports this optional precision specification,

but the precision value is used only to determine storage size. A precision from 0 to 23 results in a four-byte single-precision `FLOAT` column. A precision from 24 to 53 results in an eight-byte double-precision `DOUBLE` column.

MySQL allows a non-standard syntax: `FLOAT(M,D)` or `REAL(M,D)` or `DOUBLE PRECISION(M,D)`. Here, “(M,D)” means that values are displayed with up to *M* digits in total, of which *D* digits may be after the decimal point. For example, a column defined as `FLOAT(7,4)` will look like `-999.9999` when displayed. MySQL performs rounding when storing values, so if you insert `999.00009` into a `FLOAT(7,4)` column, the approximate result is `999.0001`.

MySQL treats `DOUBLE` as a synonym for `DOUBLE PRECISION` (a non-standard extension). MySQL also treats `REAL` as a synonym for `DOUBLE PRECISION` (a non-standard variation), unless the `REAL_AS_FLOAT` SQL mode is enabled.

For maximum portability, code requiring storage of approximate numeric data values should use `FLOAT` or `DOUBLE PRECISION` with no specification of precision or number of digits.

The `DECIMAL` and `NUMERIC` data types are used to store exact numeric data values. In MySQL, `NUMERIC` is implemented as `DECIMAL`. These types are used to store values for which it is important to preserve exact precision, for example with monetary data.

As of MySQL 5.0.3, `DECIMAL` and `NUMERIC` values are stored in binary format. Previously, they were stored as strings, with one character used for each digit of the value, the decimal point (if the scale is greater than 0), and the ‘-’ sign (for negative numbers). See [Chapter 21, Precision Math](#).

When declaring a `DECIMAL` or `NUMERIC` column, the precision and scale can be (and usually is) specified; for example:

```
salary DECIMAL(5,2)
```

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point. If the scale is 0, `DECIMAL` and `NUMERIC` values contain no decimal point or fractional part.

Standard SQL requires that the `salary` column be able to store any value with

five digits and two decimals. In this case, therefore, the range of values that can be stored in the `salary` column is from `-999.99` to `999.99`. MySQL enforces this limit as of MySQL 5.0.3. Before 5.0.3, on the positive end of the range, the column could actually store numbers up to `9999.99`. (For positive numbers, MySQL 5.0.2 and earlier used the byte reserved for the sign to extend the upper end of the range.)

In standard SQL, the syntax `DECIMAL(M)` is equivalent to `DECIMAL(M,0)`. Similarly, the syntax `DECIMAL` is equivalent to `DECIMAL(M,0)`, where the implementation is allowed to decide the value of *M*. MySQL supports both of these variant forms of the `DECIMAL` and `NUMERIC` syntax. The default value of *M* is 10.

The maximum number of digits for `DECIMAL` or `NUMERIC` is 65 (64 from MySQL 5.0.3 to 5.0.5). Before MySQL 5.0.3, the maximum range of `DECIMAL` and `NUMERIC` values is the same as for `DOUBLE`, but the actual range for a given `DECIMAL` or `NUMERIC` column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are allowed by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the allowable number of digits.)

As of MySQL 5.0.3, the `BIT` data type is used to store bit-field values. A type of `BIT(M)` allows for storage of *M*-bit values. *M* can range from 1 to 64.

To specify bit values, `b'value'` notation can be used. *value* is a binary value written using zeros and ones. For example, `b'111'` and `b'10000000'` represent 7 and 128, respectively. See [Section 9.1.5, “Bit-Field Values”](#).

If you assign a value to a `BIT(M)` column that is less than *M* bits long, the value is padded on the left with zeros. For example, assigning a value of `b'101'` to a `BIT(6)` column is, in effect, the same as assigning `b'000101'`.

When asked to store a value in a numeric column that is outside the data type's allowable range, MySQL's behavior depends on the SQL mode in effect at the time. For example, if no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the range and stores the resulting value instead. However, if the mode is set to `TRADITIONAL`, MySQL rejects a value that is out of range with an error, and the insert fails, in accordance with the SQL standard.

In non-strict mode, when an out-of-range value is assigned to an integer column,

MySQL stores the value representing the corresponding endpoint of the column data type range. If you store 256 into a `TINYINT` or `TINYINT UNSIGNED` column, MySQL stores 127 or 255, respectively. When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Conversions that occur due to clipping when MySQL is not operating in strict mode are reported as warnings for `ALTER TABLE`, `LOAD DATA INFILE`, `UPDATE`, and multiple-row `INSERT` statements. When MySQL is operating in strict mode, these statements fail, and some or all of the values will not be inserted or changed, depending on whether the table is a transactional table and other factors. For details, see [Section 5.2.5, “The Server SQL Mode”](#).

11.3. Date and Time Types

The date and time types for representing temporal values are DATETIME, DATE, TIMESTAMP, TIME, and YEAR. Each temporal type has a range of legal values, as well as a “zero” value that may be used when you specify an illegal value that MySQL cannot represent. The TIMESTAMP type has special automatic updating behavior, described later on. For temporary type storage requirements, see [Section 11.5, “Data Type Storage Requirements”](#).

Starting from MySQL 5.0.2, MySQL gives warnings or errors if you try to insert an illegal date. By setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See [Section 5.2.5, “The Server SQL Mode”](#).) You can get MySQL to accept certain dates, such as '1999-11-31', by using the ALLOW_INVALID_DATES SQL mode. (Before 5.0.2, this mode was the default behavior for MySQL.) This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 0 to 12 and that the day is in the range from 0 to 31. These ranges are defined to include zero because MySQL allows you to store dates where the day or month and day are zero in a DATE or DATETIME column. This is extremely useful for applications that need to store a birthdate for which you do not know the exact date. In this case, you simply store the date as '1999-00-00' or '1999-01-00'. If you store dates such as these, you should not expect to get correct results for functions such as DATE_SUB() or DATE_ADD that require complete dates. (If you do *not* want to allow zero in dates, you can use the NO_ZERO_IN_DATE SQL mode).

MySQL also allows you to store '0000-00-00' as a “dummy date” (if you are not using the NO_ZERO_DATE SQL mode). This is in some cases is more convenient (and uses less space in data and index) than using NULL values.

Here are some general considerations to keep in mind when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). Only the formats described in the

following sections are supported. It is expected that you supply legal values. Unpredictable results may occur if you use values in other formats.

- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using the following rules:
 - Year values in the range 70-99 are converted to 1970-1999.
 - Year values in the range 00-69 are converted to 2000-2069.
- Although MySQL tries to interpret values in several formats, dates always must be given in year-month-day order (for example, '98-09-04'), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, '09-04-98', '04-09-98').
- MySQL automatically converts a date or time type value to a number if the value is used in a numeric context and vice versa.
- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise illegal for the type (as described at the beginning of this section), it converts the value to the “zero” value for that type. The exception is that out-of-range TIME values are clipped to the appropriate endpoint of the TIME range.

The following table shows the format of the “zero” value for each type. Note that the use of these values produces warnings if the NO_ZERO_DATE SQL mode is enabled.

Data Type	“Zero” Value
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000

- The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values '0' or 0, which are easier to write.

- “Zero” date or time values used through MyODBC are converted automatically to NULL in MyODBC 2.50.12 and above, because ODBC cannot handle such values.

11.3.1. The DATETIME, DATE, and TIMESTAMP Types

The DATETIME, DATE, and TIMESTAMP types are related. This section describes their characteristics, how they are similar, and how they differ.

The DATETIME type is used when you need values that contain both date and time information. MySQL retrieves and displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

The DATE type is used when you need only a date value, without a time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.

For the DATETIME and DATE range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The TIMESTAMP data type has varying properties, depending on the MySQL version and the SQL mode the server is running in. These properties are described later in this section.

You can specify DATETIME, DATE, and TIMESTAMP values using any of a common set of formats:

- As a string in either 'YYYY-MM-DD HH:MM:SS' or 'YY-MM-DD HH:MM:SS' format. A “relaxed” syntax is allowed: Any punctuation character may be used as the delimiter between date parts or time parts. For example, '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11*30*45', and '98@12@31 11^30^45' are equivalent.
- As a string in either 'YYYY-MM-DD' or 'YY-MM-DD' format. A “relaxed” syntax is allowed here, too. For example, '98-12-31', '98.12.31', '98/12/31', and '98@12@31' are equivalent.
- As a string with no delimiters in either 'YYYYMMDDHHMMSS' or 'YYMMDDHHMMSS' format, provided that the string makes sense as a date. For

example, '19970523091528' and '970523091528' are interpreted as '1997-05-23 09:15:28', but '971122129015' is illegal (it has a nonsensical minute part) and becomes '0000-00-00 00:00:00'.

- As a string with no delimiters in either 'YYYYMMDD' or 'YYMMDD' format, provided that the string makes sense as a date. For example, '19970523' and '970523' are interpreted as '1997-05-23', but '971332' is illegal (it has nonsensical month and day parts) and becomes '0000-00-00'.
- As a number in either YYYYMMDDHHMMSS or YYMMDDHHMMSS format, provided that the number makes sense as a date. For example, 19830905132800 and 830905132800 are interpreted as '1983-09-05 13:28:00'.
- As a number in either YYYYMMDD or YYMMDD format, provided that the number makes sense as a date. For example, 19830905 and 830905 are interpreted as '1983-09-05'.
- As the result of a function that returns a value that is acceptable in a DATETIME, DATE, or TIMESTAMP context, such as NOW() or CURRENT_DATE.

Illegal DATETIME, DATE, or TIMESTAMP values are converted to the “zero” value of the appropriate type ('0000-00-00 00:00:00' or '0000-00-00').

For values specified as strings that include date part delimiters, it is not necessary to specify two digits for month or day values that are less than 10. '1979-6-9' is the same as '1979-06-09'. Similarly, for values specified as strings that include time part delimiters, it is not necessary to specify two digits for hour, minute, or second values that are less than 10. '1979-10-30 1:2:3' is the same as '1979-10-30 01:02:03'.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in YYYYMMDD or YYYYMMDDHHMMSS format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in YYMMDD or YYMMDDHHMMSS format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as non-delimited strings are interpreted using their length as given. If the string is 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2

characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify '9903', thinking that represents March, 1999, MySQL inserts a “zero” date value into your table. This occurs because the year and month values are 99 and 03, but the day part is completely missing, so the value is not a legal date. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, you can use '990300' to insert the value '1999-03-00'.

You can to some extent assign values of one date type to an object of a different date type. However, there may be some alteration of the value or loss of information:

- If you assign a DATE value to a DATETIME or TIMESTAMP object, the time part of the resulting value is set to '00:00:00' because the DATE value contains no time information.
- If you assign a DATETIME or TIMESTAMP value to a DATE object, the time part of the resulting value is deleted because the DATE type stores no time information.
- Remember that although DATETIME, DATE, and TIMESTAMP values all can be specified using the same set of formats, the types do not all have the same range of values. For example, TIMESTAMP values cannot be earlier than 1970 or later than 2037. This means that a date such as '1968-01-01', while legal as a DATETIME or DATE value, is not valid as a TIMESTAMP value and is converted to 0.

Be aware of certain pitfalls when specifying date values:

- The relaxed format allowed for values specified as strings can be deceiving. For example, a value such as '10:11:12' might look like a time value because of the ':' delimiter, but if used in a date context is interpreted as the year '2010-11-12'. The value '10:45:15' is converted to '0000-00-00' because '45' is not a legal month.
- As of 5.0.2, the server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as '2004-04-31' are converted to '0000-00-

00' and a warning is generated. With strict mode enabled, invalid dates generate an error. To allow such dates, enable `ALLOW_INVALID_DATES`. See [Section 5.2.5, “The Server SQL Mode”](#), for more information.

Before MySQL 5.0.2, the MySQL server performs only basic checking on the validity of a date: The ranges for year, month, and day are 1000 to 9999, 00 to 12, and 00 to 31, respectively. Any date containing parts not within these ranges is subject to conversion to '0000-00-00'. Please note that this still allows you to store invalid dates such as '2002-04-31'. To ensure that a date is valid, you should perform a check in your application.

- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using the following rules:
 - Year values in the range 00-69 are converted to 2000-2069.
 - Year values in the range 70-99 are converted to 1970-1999.

11.3.1.1. **TIMESTAMP Properties as of MySQL 4.1**

Note: In older versions of MySQL (prior to 4.1), the properties of the `TIMESTAMP` data type differed significantly in many ways from what is described in this section. If you need to convert older `TIMESTAMP` data to work with MySQL 5.0, be sure to see the *MySQL 3.23, 4.0, 4.1 Reference Manual* for details.

`TIMESTAMP` columns are displayed in the same format as `DATETIME` columns. In other words, the display width is fixed at 19 characters, and the format is `YYYY-MM-DD HH:MM:SS`.

The MySQL server can be also be run with the `MAXDB SQL` mode enabled. When the server runs with this mode enabled, `TIMESTAMP` is identical with `DATETIME`. That is, if this mode is enabled at the time that a table is created, `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and there is no automatic initialization or updating to the current date and time.

To enable `MAXDB` mode, set the server SQL mode to `MAXDB` at startup using the `--sql-mode=MAXDB` server option or by setting the global `sql_mode` variable at

runtime:

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

A client can cause the server to run in MAXDB mode for its own connection as follows:

```
mysql> SET SESSION sql_mode=MAXDB;
```

Note that the information in the following discussion applies to `TIMESTAMP` columns only for tables not created with MAXDB mode enabled, because such columns are created as `DATETIME` columns.

As of MySQL 5.0.2, MySQL does not accept timestamp values that include a zero in the day or month column or values that are not a valid date. The sole exception to this rule is the special value `'0000-00-00 00:00:00'`.

You have considerable flexibility in determining when automatic `TIMESTAMP` initialization and updating occur and which column should have those behaviors:

- For one `TIMESTAMP` column in a table, you can assign the current timestamp as the default value and the auto-update value. It is possible to have the current timestamp be the default value for initializing the column, for the auto-update value, or both. It is not possible to have the current timestamp be the default value for one column and the auto-update value for another column.
- You can specify which `TIMESTAMP` column to automatically initialize or update to the current date and time. This need not be the first `TIMESTAMP` column.

The following rules govern initialization and updating of `TIMESTAMP` columns:

- If a `DEFAULT` value is specified for the first `TIMESTAMP` column in a table, it is not ignored. The default can be `CURRENT_TIMESTAMP` or a constant date and time value.
- `DEFAULT NULL` is the same as `DEFAULT CURRENT_TIMESTAMP` for the *first* `TIMESTAMP` column. For any other `TIMESTAMP` column, `DEFAULT NULL` is treated as `DEFAULT 0`.

- Any single `TIMESTAMP` column in a table can be used as the one that is initialized to the current timestamp or updated automatically.
- In a `CREATE TABLE` statement, the first `TIMESTAMP` column can be declared in any of the following ways:
 - With both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses, the column has the current timestamp for its default value, and is automatically updated.
 - With neither `DEFAULT` nor `ON UPDATE` clauses, it is the same as `DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP`.
 - With a `DEFAULT CURRENT_TIMESTAMP` clause and no `ON UPDATE` clause, the column has the current timestamp for its default value but is not automatically updated.
 - With no `DEFAULT` clause and with an `ON UPDATE CURRENT_TIMESTAMP` clause, the column has a default of 0 and is automatically updated.
 - With a constant `DEFAULT` value, the column has the given default. If the column has an `ON UPDATE CURRENT_TIMESTAMP` clause, it is automatically updated, otherwise not.

In other words, you can use the current timestamp for both the initial value and the auto-update value, or either one, or neither. (For example, you can specify `ON UPDATE` to enable auto-update without also having the column auto-initialized.)

- `CURRENT_TIMESTAMP` or any of its synonyms (`CURRENT_TIMESTAMP()`, `NOW()`, `LOCALTIME`, `LOCALTIME()`, `LOCALTIMESTAMP`, or `LOCALTIMESTAMP()`) can be used in the `DEFAULT` and `ON UPDATE` clauses. They all mean “the current timestamp.” (`UTC_TIMESTAMP` is not allowed. Its range of values does not align with those of the `TIMESTAMP` column anyway unless the current time zone is UTC.)
- The order of the `DEFAULT` and `ON UPDATE` attributes does not matter. If both `DEFAULT` and `ON UPDATE` are specified for a `TIMESTAMP` column, either can precede the other. For example, these statements are equivalent:

```

CREATE TABLE t (ts TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                 ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
                 DEFAULT CURRENT_TIMESTAMP);

```

- To specify automatic default or updating for a `TIMESTAMP` column other than the first one, you must suppress the automatic initialization and update behaviors for the first `TIMESTAMP` column by explicitly assigning it a constant `DEFAULT` value (for example, `DEFAULT 0` or `DEFAULT '2003-01-01 00:00:00'`). Then, for the other `TIMESTAMP` column, the rules are the same as for the first `TIMESTAMP` column, except that if you omit both of the `DEFAULT` and `ON UPDATE` clauses, no automatic initialization or updating occurs.

Example. These statements are equivalent:

```

CREATE TABLE t (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
        DEFAULT CURRENT_TIMESTAMP);

```

You can set the current time zone on a per-connection basis, as described in [Section 5.11.8, “MySQL Server Time Zone Support”](#). `TIMESTAMP` values are stored in UTC, being converted from the current time zone for storage, and converted back to the current time zone upon retrieval. As long as the time zone setting remains constant, you get back the same value you store. If you store a `TIMESTAMP` value, and then change the time zone and retrieve the value, the retrieved value is different than the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the `time_zone` system variable.

You can include the `NULL` attribute in the definition of a `TIMESTAMP` column to allow the column to contain `NULL` values. For example:

```

CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,

```

```
ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

If the NULL attribute is not specified, setting the column to NULL sets it to the current timestamp. Note that a TIMESTAMP column which allows NULL values will *not* take on the current timestamp except under one of the following conditions:

- Its default value is defined as CURRENT_TIMESTAMP
- NOW() or CURRENT_TIMESTAMP is inserted into the column

In other words, a TIMESTAMP column defined as NULL will auto-initialize only if it is created using a definition such as the following:

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```

Otherwise — that is, if the TIMESTAMP column is defined to allow NULL values but not using DEFAULT TIMESTAMP, as shown here...

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT NULL);
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
```

...then you must explicitly insert a value corresponding to the current date and time. For example:

```
INSERT INTO t1 VALUES (NOW());
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

11.3.2. The TIME Type

MySQL retrieves and displays TIME values in 'HH:MM:SS' format (or 'HHH:MM:SS' format for large hours values). TIME values may range from '-838:59:59' to '838:59:59'. The hours part may be so large because the TIME type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

You can specify TIME values in a variety of formats:

- As a string in 'D HH:MM:SS.fraction' format. You can also use one of the following “relaxed” syntaxes: 'HH:MM:SS.fraction', 'HH:MM:SS',

'HH:MM', 'D HH:MM:SS', 'D HH:MM', 'D HH', or 'SS'. Here D represents days and can have a value from 0 to 34. Note that MySQL does not store the fraction part.

- As a string with no delimiters in 'HHMMSS' format, provided that it makes sense as a time. For example, '101112' is understood as '10:11:12', but '109712' is illegal (it has a nonsensical minute part) and becomes '00:00:00'.
- As a number in HHMMSS format, provided that it makes sense as a time. For example, 101112 is understood as '10:11:12'. The following alternative formats are also understood: SS, MMSS, HHMMSS, HHMMSS.fraction. Note that MySQL does not store the fraction part.
- As the result of a function that returns a value that is acceptable in a TIME context, such as CURRENT_TIME.

For TIME values specified as strings that include a time part delimiter, it is not necessary to specify two digits for hours, minutes, or seconds values that are less than 10. '8:3:2' is the same as '08:03:02'.

Be careful about assigning abbreviated values to a TIME column. Without colons, MySQL interprets values using the assumption that the two rightmost digits represent seconds. (MySQL interprets TIME values as elapsed time rather than as time of day.) For example, you might think of '1112' and 1112 as meaning '11:12:00' (12 minutes after 11 o'clock), but MySQL interprets them as '00:11:12' (11 minutes, 12 seconds). Similarly, '12' and 12 are interpreted as '00:00:12'. TIME values with colons, by contrast, are always treated as time of the day. That is, '11:12' mean '11:12:00', not '00:11:12'.

By default, values that lie outside the TIME range but are otherwise legal are clipped to the closest endpoint of the range. For example, '-850:00:00' and '850:00:00' are converted to '-838:59:59' and '838:59:59'. Illegal TIME values are converted to '00:00:00'. Note that because '00:00:00' is itself a legal TIME value, there is no way to tell, from a value of '00:00:00' stored in a table, whether the original value was specified as '00:00:00' or whether it was illegal.

For more restrictive treatment of invalid TIME values, enable strict SQL mode to cause errors to occur. See [Section 5.2.5, “The Server SQL Mode”](#).

11.3.3. The YEAR Type

The YEAR type is a one-byte type used for representing years.

MySQL retrieves and displays YEAR values in YYYY format. The range is 1901 to 2155.

You can specify YEAR values in a variety of formats:

- As a four-digit string in the range '1901' to '2155'.
- As a four-digit number in the range 1901 to 2155.
- As a two-digit string in the range '00' to '99'. Values in the ranges '00' to '69' and '70' to '99' are converted to YEAR values in the ranges 2000 to 2069 and 1970 to 1999.
- As a two-digit number in the range 1 to 99. Values in the ranges 1 to 69 and 70 to 99 are converted to YEAR values in the ranges 2001 to 2069 and 1970 to 1999. Note that the range for two-digit numbers is slightly different from the range for two-digit strings, because you cannot specify zero directly as a number and have it be interpreted as 2000. You must specify it as a string '0' or '00' or it is interpreted as 0000.
- As the result of a function that returns a value that is acceptable in a YEAR context, such as NOW().

Illegal YEAR values are converted to 0000.

11.3.4. Y2K Issues and Date Types

As discussed in [Section 1.4.5, “Year 2000 Compliance”](#), MySQL itself is year 2000 (Y2K) safe. However, particular input values presented to MySQL may not be Y2K safe. Any value containing a two-digit year is ambiguous, because the century is unknown. Such values must be interpreted into four-digit form because MySQL stores years internally using four digits.

For DATETIME, DATE, TIMESTAMP, and YEAR types, MySQL interprets dates with ambiguous year values using the following rules:

- Year values in the range 00-69 are converted to 2000-2069.
- Year values in the range 70-99 are converted to 1970-1999.

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the correct values, you should provide unambiguous input containing four-digit year values.

`ORDER BY` properly sorts `YEAR` values that have two-digit years.

Some functions like `MIN()` and `MAX()` convert a `YEAR` to a number. This means that a value with a two-digit year does not work properly with these functions. The fix in this case is to convert the `TIMESTAMP` or `YEAR` to four-digit year format.

11.4. String Types

The string types are CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, and SET. This section describes how these types work and how to use them in your queries. For string type storage requirements, see [Section 11.5, “Data Type Storage Requirements”](#).

11.4.1. The CHAR and VARCHAR Types

The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved. As of MySQL 5.0.3, they also differ in maximum length and in whether trailing spaces are retained.

The CHAR and VARCHAR types are declared with a length that indicates the maximum number of characters you want to store. For example, CHAR(30) can hold up to 30 characters.

The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed.

Values in VARCHAR columns are variable-length strings. The length can be specified as a value from 0 to 255 before MySQL 5.0.3, and 0 to 65,535 in 5.0.3 and later versions. (The maximum effective length of a VARCHAR in MySQL 5.0.3 and later is determined by the maximum row size and the character set used. The maximum length overall is 65,532 bytes.)

In contrast to CHAR, VARCHAR values are stored using only as many characters as are needed, plus one byte to record the length (two bytes for columns that are declared with a length longer than 255).

VARCHAR values are not padded when they are stored. Handling of trailing spaces is version-dependent. As of MySQL 5.0.3, trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL. Before MySQL 5.0.3, trailing spaces are removed from values when they are stored into a VARCHAR column; this means that the spaces also are absent from retrieved values.

If you assign a value to a CHAR or VARCHAR column that exceeds the column's maximum length, the value is truncated to fit. If the truncated characters are not spaces, a warning is generated. For truncation of non-space characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See [Section 5.2.5, “The Server SQL Mode”](#).

Before MySQL 5.0.3, if you need a data type for which trailing spaces are not removed, consider using a BLOB or TEXT type. Also, if you want to store binary values such as results from an encryption or compression function that might contain arbitrary byte values, use a BLOB column rather than a CHAR or VARCHAR column, to avoid potential problems with trailing space removal that would change data values.

The following table illustrates the differences between CHAR and VARCHAR by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns:

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
' '	' '	4 bytes	' '	1 byte
'ab '	'ab '	4 bytes	'ab '	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Note that the values shown as stored in the last row of the table apply *only when not using strict mode*; if MySQL is running in strict mode, values that exceed the column length are *not stored*, and an error results.

If a given value is stored into the CHAR(4) and VARCHAR(4) columns, the values retrieved from the columns are not always the same because trailing spaces are removed from CHAR columns upon retrieval. The following example illustrates this difference:

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO vc VALUES ('ab ', 'ab ');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT CONCAT('(' , v, ')'), CONCAT('(' , c, ')') FROM vc;
```

```
+-----+-----+
```

```

| CONCAT('( ', v, ' ') | CONCAT('( ', c, ' ') |
+-----+-----+
| (ab )                | (ab)                |
+-----+-----+
1 row in set (0.06 sec)

```

Values in CHAR and VARCHAR columns are sorted and compared according to the character set collation assigned to the column.

Note that all MySQL collations are of type PADSPACE. This means that all CHAR and VARCHAR values in MySQL are compared without regard to any trailing spaces. For example:

```
mysql> CREATE TABLE names (myname CHAR(10), yourname VARCHAR(10));
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> INSERT INTO names VALUES ('Monty ', 'Monty ');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT myname = 'Monty ', yourname = 'Monty ' FROM names;
+-----+-----+
| myname = 'Monty ' | yourname = 'Monty ' |
+-----+-----+
|                  1 |                  1 |
+-----+-----+
1 row in set (0.00 sec)

```

Note that this is true for all MySQL versions, and it makes no difference whether your version trims trailing spaces from VARCHAR values before storing them. Nor does the server SQL mode make any difference in this regard.

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error.

11.4.2. The BINARY and VARBINARY Types

The BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they contain binary strings rather than non-binary strings. That is, they contain byte strings rather than character strings. This means that they have no character set, and sorting and comparison are based on the numeric values of the bytes in

the values.

The allowable maximum length is the same for `BINARY` and `VARBINARY` as it is for `CHAR` and `VARCHAR`, except that the length for `BINARY` and `VARBINARY` is a length in bytes rather than in characters.

The `BINARY` and `VARBINARY` data types are distinct from the `CHAR BINARY` and `VARCHAR BINARY` data types. For the latter types, the `BINARY` attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary collation for the column character set to be used, and the column itself contains non-binary character strings rather than binary byte strings. For example, `CHAR(5) BINARY` is treated as `CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin`, assuming that the default character set is `latin1`. This differs from `BINARY(5)`, which stores 5-bytes binary strings that have no character set or collation.

When `BINARY` values are stored, they are right-padded with the pad value to the specified length. The pad value and how it is handled is version specific:

- As of MySQL 5.0.15, the pad value is `0x00` (the zero byte). Values are right-padded with `0x00` on insert, and no trailing bytes are removed on select. All bytes are significant in comparisons, including `ORDER BY` and `DISTINCT` operations. `0x00` bytes and spaces are different in comparisons, with `0x00 < space`.

Example: For a `BINARY(3)` column, `'a '` becomes `'a \0'` when inserted. `'a\0'` becomes `'a\0\0'` when inserted. Both inserted values remain unchanged when selected.

- Before MySQL 5.0.15, the pad value is space. Values are right-padded with space on insert, and trailing spaces are removed on select. Trailing spaces are ignored in comparisons, including `ORDER BY` and `DISTINCT` operations. `0x00` bytes and spaces are different in comparisons, with `0x00 < space`.

Example: For a `BINARY(3)` column, `'a '` becomes `'a '` when inserted and `'a'` when selected. `'a\0'` becomes `'a\0 '` when inserted and `'a\0'` when selected.

For `VARBINARY`, there is no padding on insert and no bytes are stripped on select. All bytes are significant in comparisons, including `ORDER BY` and `DISTINCT`

operations. 0x00 bytes and spaces are different in comparisons, with 0x00 < space. (Exceptions: Before MySQL 5.0.3, trailing spaces are removed when values are stored. Before MySQL 5.0.15, trailing 0x00 bytes are removed for ORDER BY operations.)

Note: The InnoDB storage engine continues to preserve trailing spaces in BINARY and VARBINARY column values through MySQL 5.0.18. Beginning with MySQL 5.0.19, InnoDB uses trailing space characters in making comparisons as do other MySQL storage engines.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad bytes will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a\0' causes a duplicate-key error.

You should consider the preceding padding and stripping characteristics carefully if you plan to use the BINARY data type for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how 0x00-padding of BINARY values affects column value comparisons:

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;
+-----+-----+-----+
| HEX(c) | c = 'a' | c = 'a\0\0' |
+-----+-----+-----+
| 610000 |      0 |           1 |
+-----+-----+-----+
1 row in set (0.09 sec)
```

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use VARBINARY or one of the BLOB data types instead.

11.4.3. The BLOB and TEXT Types

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. These differ only in the maximum length of the values they can hold. The four TEXT types are TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. These correspond to the four BLOB types and have the same maximum lengths and storage requirements. See [Section 11.5, “Data Type Storage Requirements”](#). No lettercase conversion for TEXT or BLOB columns takes place during storage or retrieval.

BLOB columns are treated as binary strings (byte strings). TEXT columns are treated as non-binary strings (character strings). BLOB columns have no character set, and sorting and comparison are based on the numeric values of the bytes in column values. TEXT columns have a character set, and values are sorted and compared based on the collation of the character set.

If a TEXT column is indexed, index entry comparisons are space-padded at the end. This means that, if the index requires unique values, duplicate-key errors will occur for values that differ only in the number of trailing spaces. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error. This is not true for BLOB columns.

When not running in strict mode, if you assign a value to a BLOB or TEXT column that exceeds the data type's maximum length, the value is truncated to fit. If the truncated characters are not spaces, a warning is generated. You can cause an error to occur and the value to be rejected rather than to be truncated with a warning by using strict SQL mode. See [Section 5.2.5, “The Server SQL Mode”](#).

In most respects, you can regard a BLOB column as a VARBINARY column that can be as large as you like. Similarly, you can regard a TEXT column as a VARCHAR column. BLOB and TEXT differ from VARBINARY and VARCHAR in the following ways:

- There is no trailing-space removal for BLOB and TEXT columns when values are stored or retrieved. Before MySQL 5.0.3, this differs from VARBINARY and VARCHAR, for which trailing spaces are removed when values are stored.

Note that TEXT is on comparison space extended to fit the compared object, exactly like CHAR and VARCHAR.

- For indexes on BLOB and TEXT columns, you must specify an index prefix length. For CHAR and VARCHAR, a prefix length is optional. See [Section 7.4.3](#),

[“Column Indexes”](#).

- BLOB and TEXT columns cannot have DEFAULT values.

LONG and LONG VARCHAR map to the MEDIUMTEXT data type. This is a compatibility feature. If you use the BINARY attribute with a TEXT data type, the column is assigned the binary collation of the column character set.

MySQL Connector/ODBC defines BLOB values as LONGVARBINARY and TEXT values as LONGVARCHAR.

Because BLOB and TEXT values can be extremely long, you might encounter some constraints in using them:

- Only the first `max_sort_length` bytes of the column are used when sorting. The default value of `max_sort_length` is 1024. This value can be changed using the `--max_sort_length=N` option when starting the **mysqld** server. See [Section 5.2.2, “Server System Variables”](#).

You can make more bytes significant in sorting or grouping by increasing the value of `max_sort_length` at runtime. Any client can change the value of its session `max_sort_length` variable:

```
mysql> SET max_sort_length = 2000;
mysql> SELECT id, comment FROM t
      -> ORDER BY comment;
```

Another way to use GROUP BY or ORDER BY on a BLOB or TEXT column containing long values when you want more than `max_sort_length` bytes to be significant is to convert the column value into a fixed-length object. The standard way to do this is with the SUBSTRING() function. For example, the following statement causes 2000 bytes of the comment column to be taken into account for sorting:

```
mysql> SELECT id, SUBSTRING(comment,1,2000) FROM t
      -> ORDER BY SUBSTRING(comment,1,2000);
```

- The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by

changing the value of the `max_allowed_packet` variable, but you must do so for both the server and your client program. For example, both **mysql** and **mysqldump** allow you to change the client-side `max_allowed_packet` value. See [Section 7.5.2, “Tuning Server Parameters”](#), [Section 8.6, “mysql — The MySQL Command-Line Tool”](#), and [Section 8.12, “mysqldump — A Database Backup Program”](#).

Each BLOB or TEXT value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

In some cases, it may be desirable to store binary data such as media files in BLOB or TEXT columns. You may find MySQL's string handling functions useful for working with such data. See [Section 12.3, “String Functions”](#). For security and other reasons, it is usually preferable to do so using application code rather than allowing application users the FILE privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (<http://forums.mysql.com/>).

11.4.4. The ENUM Type

An ENUM is a string object with a value chosen from a list of allowed values that are enumerated explicitly in the column specification at table creation time.

An enumeration value must be a quoted string literal; it may not be an expression, even one that evaluates to a string value. This means that you also may not employ a user variable as an enumeration value.

For example, you can create a table with an ENUM column like this:

```
CREATE TABLE sizes (  
    name ENUM('small', 'medium', 'large')  
);
```

However, this version of the previous CREATE TABLE statement does *not* work:

```
CREATE TABLE sizes (  
    c1 ENUM('small', CONCAT('med', 'ium')), 'large')  
);
```

You also may not employ a user variable as an enumeration value. This pair of

statements do *not* work:

```
SET @mysize = 'medium';
```

```
CREATE TABLE sizes (  
    name ENUM('small', @mysize, 'large')  
);
```

If you wish to use a number as an enumeration value, you must enclose it in quotes.

The value may also be the empty string (' ') or NULL under certain circumstances:

- If you insert an invalid value into an ENUM (that is, a string not present in the list of allowed values), the empty string is inserted instead as a special error value. This string can be distinguished from a “normal” empty string by the fact that this string has the numerical value 0. More about this later.

If strict SQL mode is enabled, attempts to insert invalid ENUM values result in an error.

- If an ENUM column is declared to allow NULL, the NULL value is a legal value for the column, and the default value is NULL. If an ENUM column is declared NOT NULL, its default value is the first element of the list of allowed values.

Each enumeration value has an index:

- Values from the list of allowable elements in the column specification are numbered beginning with 1.
- The index value of the empty string error value is 0. This means that you can use the following SELECT statement to find rows into which invalid ENUM values were assigned:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- The index of the NULL value is NULL.
- The term “index” here refers only to position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as `ENUM('one', 'two', 'three')` can have any of the values shown here. The index of each value is also shown:

Value	Index
NULL	NULL
' '	0
'one'	1
'two'	2
'three'	3

An enumeration can have a maximum of 65,535 elements.

Trailing spaces are automatically deleted from `ENUM` member values in the table definition when a table is created.

When retrieved, values stored into an `ENUM` column are displayed using the lettercase that was used in the column definition. Note that `ENUM` columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

If you retrieve an `ENUM` value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an `ENUM` column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

If you store a number into an `ENUM` column, the number is treated as an index, and the value stored is the enumeration member with that index. (However, this does not work with `LOAD DATA`, which treats all input as strings.) It is not advisable to define an `ENUM` column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of `'0'`, `'1'`, and `'2'`, but numeric index values of 1, 2, and 3:

```
numbers ENUM('0', '1', '2')
```

`ENUM` values are sorted according to the order in which the enumeration members were listed in the column specification. (In other words, `ENUM` values are sorted according to their index numbers.) For example, `'a'` sorts before `'b'` for

ENUM('a', 'b'), but 'b' sorts before 'a' for ENUM('b', 'a'). The empty string sorts before non-empty strings, and NULL values sort before all other enumeration values. To prevent unexpected results, specify the ENUM list in alphabetical order. You can also use GROUP BY CAST(col AS CHAR) or GROUP BY CONCAT(col) to make sure that the column is sorted lexically rather than by index number.

If you want to determine all possible values for an ENUM column, use SHOW COLUMNS FROM tbl_name LIKE enum_col and parse the ENUM definition in the Type column of the output.

11.4.5. The SET Type

A SET is a string object that can have zero or more values, each of which must be chosen from a list of allowed values specified when the table is created. SET column values that consist of multiple set members are specified with members separated by commas (','). A consequence of this is that SET member values should not themselves contain commas.

For example, a column specified as SET('one', 'two') NOT NULL can have any of these values:

```
' '  
'one '  
'two '  
'one, two '
```

A SET can have a maximum of 64 different members.

Trailing spaces are automatically deleted from SET member values in the table definition when a table is created.

When retrieved, values stored in a SET column are displayed using the lettercase that was used in the column definition. Note that SET columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

MySQL stores SET values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a SET value in a numeric context, the value retrieved has bits set corresponding to the set members that

make up the column value. For example, you can retrieve numeric values from a SET column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as SET('a', 'b', 'c', 'd'), the members have the following decimal and binary values:

SET Member	Decimal Value	Binary Value
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. For example, suppose that a column is specified as SET('a', 'b', 'c', 'd'):

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES  
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');  
Query OK, 5 rows affected (0.01 sec)  
Records: 5 Duplicates: 0 Warnings: 0
```

Then all of these values appear as 'a,d' when retrieved:

```
mysql> SELECT col FROM myset;  
+-----+  
| col   |  
+-----+
```

```

| a,d |
+-----+
5 rows in set (0.04 sec)

```

If you set a SET column to an unsupported value, the value is ignored and a warning is issued:

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)
```

```
mysql> SHOW WARNINGS;
```

```

+-----+-----+-----+-----+
| Level   | Code | Message                                     |
+-----+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+-----+-----+-----+-----+
1 row in set (0.04 sec)

```

```
mysql> SELECT col FROM myset;
```

```

+-----+
| col   |
+-----+
| a,d   |
+-----+
6 rows in set (0.01 sec)

```

If strict SQL mode is enabled, attempts to insert invalid SET values result in an error.

SET values are sorted numerically. NULL values sort before non-NULL SET values.

Normally, you search for SET values using the `FIND_IN_SET()` function or the `LIKE` operator:

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

The first statement finds rows where `set_col` contains the `value` set member.

The second is similar, but not the same: It finds rows where *set_col* contains *value* anywhere, even as a substring of another set member.

The following statements also are legal:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;  
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to 'val1,val2' returns different results than comparing values to 'val2,val1'. You should specify the values in the same order they are listed in the column definition.

If you want to determine all possible values for a SET column, use `SHOW COLUMNS FROM tbl_name LIKE set_col` and parse the SET definition in the `TYPE` column of the output.

11.5. Data Type Storage Requirements

The storage requirements for each of the data types supported by MySQL are listed here by category.

The maximum size of a row in a MyISAM table is 65,534 bytes. Each BLOB and TEXT column accounts for only five to nine bytes toward this size.

Important: For tables using the NDBCluster storage engine, there is the factor of *4-byte alignment* to be taken into account when calculating storage requirements. This means that all NDB data storage is done in multiples of 4 bytes. Thus, a column value that — in a table using a storage engine other than NDB — would take 15 bytes for storage, requires 16 bytes in an NDB table. This requirement applies in addition to any other considerations that are discussed in this section. For example, in NDBCluster tables, the TINYINT, SMALLINT, MEDIUMINT, and INTEGER (INT) column types each require 4 bytes storage per record.

In addition, when calculating storage requirements for Cluster tables, you must remember that every table using the NDBCluster storage engine requires a primary key; if no primary key is defined by the user, then a “hidden” primary key will be created by NDB. This hidden primary key consumes 31-35 bytes per table record.

When calculating Cluster memory requirements, you may find useful the `ndb_size.pl` utility which is available on [MySQLForge](#). This Perl script connects to a current MySQL (non-Cluster) database and creates a report on how much space that database would require if it used the NDBCluster storage engine.

Storage Requirements for Numeric Types

Data Type	Storage Required
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT, INTEGER	4 bytes
BIGINT	

	8 bytes
FLOAT(<i>p</i>)	4 bytes if $0 \leq p \leq 24$, 8 bytes if $25 \leq p \leq 53$
FLOAT	4 bytes
DOUBLE [PRECISION], REAL	8 bytes
DECIMAL(<i>M,D</i>), NUMERIC(<i>M,D</i>)	Varies; see following discussion
BIT(<i>M</i>)	approximately $(M+7)/8$ bytes

The storage requirements for DECIMAL (and NUMERIC) are version-specific:

As of MySQL 5.0.3, values for DECIMAL columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table:

Leftover Digits	Number of Bytes
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	4

Before MySQL 5.0.3, DECIMAL columns are represented as strings and storage requirements are: $M+2$ bytes if $D > 0$, $M+1$ bytes if $D = 0$ ($D+2$, if $M < D$)

Storage Requirements for Date and Time Types

Data Type	Storage Required
-----------	------------------

DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

Storage Requirements for String Types

Data Type	Storage Required
CHAR(M)	M bytes, $0 \leq M \leq 255$
VARCHAR(M)	<i>Prior to MySQL 5.0.3: L + 1 bytes, where $L \leq M$ and $0 \leq M \leq 255$. MySQL 5.0.3 and later: L + 1 bytes, where $L \leq M$ and $0 \leq M \leq 255$ or L + 2 bytes, where $L \leq M$ and $256 \leq M \leq 65535$ (see note below).</i>
BINARY(M)	M bytes, $0 \leq M \leq 255$
VARBINARY(M)	<i>Prior to MySQL 5.0.3: L + 1 bytes, where $L \leq M$ and $0 \leq M \leq 255$. MySQL 5.0.3 and later: L + 1 bytes, where $L \leq M$ and $0 \leq M \leq 255$ or L + 2 bytes, where $L \leq M$ and $256 \leq M \leq 65535$ (see note below).</i>
TINYBLOB, TINYTEXT	$L+1$ byte, where $L < 2^8$
BLOB, TEXT	$L+2$ bytes, where $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	$L+3$ bytes, where $L < 2^{24}$
LOBLOB, LONGTEXT	$L+4$ bytes, where $L < 2^{32}$
ENUM('value1','value2',...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET('value1','value2',...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

For the CHAR, VARCHAR, and TEXT types, the values L and M in the preceding table should be interpreted as number of characters, and lengths for these types in column specifications indicate the number of characters. For example, to store a TINYTEXT value requires L characters plus one byte.

VARCHAR, VARBINARY, and the BLOB and TEXT types are variable-length types. For each, the storage requirements depend on these factors:

- The actual length of the column value
- The column's maximum possible length
- The character set used for the column

For example, a VARCHAR(10) column can hold a string with a maximum length of 10. Assuming that the column uses the latin1 character set (one byte per character), the actual storage required is the length of the string (L), plus one byte to record the length of the string. For the string 'abcd', L is 4 and the storage requirement is five bytes. If the same column was instead declared as VARCHAR(500), the string 'abcd' requires $4 + 2 = 6$ bytes. Two bytes rather than one are required for the prefix because the length of the column is greater than 255 characters.

To calculate the number of *bytes* used to store a particular CHAR, VARCHAR, or TEXT column value, you must take into account the character set used for that column. In particular, when using the utf8 Unicode character set, you must keep in mind that not all utf8 characters use the same number of bytes. For a breakdown of the storage used for different categories of utf8 characters, see [Section 10.7, “Unicode Support”](#).

Note: In MySQL 5.0.3 and later, the *effective* maximum length for a VARCHAR or VARBINARY column is 65,532.

As of MySQL 5.0.3, the NDBCLUSTER engine supports only fixed-width columns. This means that a VARCHAR column from a table in a MySQL Cluster will behave as follows:

- If the size of the column is fewer than 256 characters, the column requires one byte extra storage per row.
- If the size of the column is 256 characters or more, the column requires two bytes extra storage per row.

Note that the number of bytes required per character varies according to the character set used. For example, if a VARCHAR(100) column in a Cluster table

uses the `utf8` character set, then each character requires 3 bytes storage. This means that each record in such a column takes up $100 \times 3 + 1 = 301$ bytes for storage, regardless of the length of the string actually stored in any given record. For a `VARCHAR(1000)` column in a table using the `NDBCLUSTER` storage engine with the `utf8` character set, each record will use $1000 \times 3 + 2 = 3002$ bytes storage; that is, the column is 1,000 characters wide, each character requires 3 bytes storage, and each record has a 2-byte overhead because $1,000 > 256$.

The `BLOB` and `TEXT` types require 1, 2, 3, or 4 bytes to record the length of the column value, depending on the maximum possible length of the type. See [Section 11.4.3, “The BLOB and TEXT Types”](#).

`TEXT` and `BLOB` columns are implemented differently in the `NDB Cluster` storage engine, wherein each row in a `TEXT` column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which stored in a hidden table. The rows in this second table are always 2,000 bytes long. This means that the size of a `TEXT` column is 256 if $size \leq 256$ (where $size$ represents the size of the row); otherwise, the size is $256 + size + (2000 - (size - 256) \% 2000)$.

The size of an `ENUM` object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See [Section 11.4.4, “The ENUM Type”](#).

The size of a `SET` object is determined by the number of different set members. If the set size is N , the object occupies $(N+7)/8$ bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A `SET` can have a maximum of 64 members. See [Section 11.4.5, “The SET Type”](#).

11.6. Choosing the Right Type for a Column

For optimum storage, you should try to use the most precise type in all cases. For example, if an integer column is used for values in the range from 1 to 99999, `MEDIUMINT UNSIGNED` is the best type. Of the types that represent all the required values, this type uses the least amount of storage.

Tables created in MySQL 5.0.3 and above uses a new storage format for `DECIMAL` columns. All basic calculation (+, -, *, /) with `DECIMAL` columns are done with precision of 65 decimal (base 10) digits. See [Section 11.1.1, “Overview of Numeric Types”](#).

Prior to MySQL 5.0.3, calculations on `DECIMAL` values are performed using double-precision operations. If accuracy is not too important or if speed is the highest priority, the `DOUBLE` type may be good enough. For high precision, you can always convert to a fixed-point type stored in a `BIGINT`. This allows you to do all calculations with 64-bit integers and then convert results back to floating-point values as necessary.

11.7. Using Data Types from Other Database Engines

To facilitate the use of code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL:

Other Vendor Type	MySQL Type
BOOL,	TINYINT
BOOLEAN	TINYINT
CHAR VARYING(M)	VARCHAR(M)
DEC	DECIMAL
FIXED	DECIMAL
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a `DESCRIBE tbl_name` statement, MySQL reports the table structure using the equivalent MySQL types. For example:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DESCRIBE t;
```

```
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
```

a	tinyint(1)	YES	NULL
b	double	YES	NULL
c	mediumtext	YES	NULL
d	decimal(10,0)	YES	NULL

4 rows in set (0.01 sec)

Chapter 12. Functions and Operators

Table of Contents

[12.1. Operators](#)

[12.1.1. Operator Precedence](#)

[12.1.2. Type Conversion in Expression Evaluation](#)

[12.1.3. Comparison Functions and Operators](#)

[12.1.4. Logical Operators](#)

[12.2. Control Flow Functions](#)

[12.3. String Functions](#)

[12.3.1. String Comparison Functions](#)

[12.4. Numeric Functions](#)

[12.4.1. Arithmetic Operators](#)

[12.4.2. Mathematical Functions](#)

[12.5. Date and Time Functions](#)

[12.6. What Calendar Is Used By MySQL?](#)

[12.7. Full-Text Search Functions](#)

[12.7.1. Boolean Full-Text Searches](#)

[12.7.2. Full-Text Searches with Query Expansion](#)

[12.7.3. Full-Text Stopwords](#)

[12.7.4. Full-Text Restrictions](#)

[12.7.5. Fine-Tuning MySQL Full-Text Search](#)

[12.8. Cast Functions and Operators](#)

[12.9. Other Functions](#)

[12.9.1. Bit Functions](#)

[12.9.2. Encryption and Compression Functions](#)

[12.9.3. Information Functions](#)

[12.9.4. Miscellaneous Functions](#)

[12.10. Functions and Modifiers for Use with GROUP BY Clauses](#)

[12.10.1. GROUP BY \(Aggregate\) Functions](#)

[12.10.2. GROUP BY Modifiers](#)

[12.10.3. GROUP BY and HAVING with Hidden Fields](#)

Expressions can be used at several points in SQL statements, such as in the ORDER BY or HAVING clauses of SELECT statements, in the WHERE clause of a SELECT, DELETE, or UPDATE statement, or in SET statements. Expressions can be

written using literal values, column values, NULL, built-in functions, stored functions, user-defined functions, and operators. This chapter describes the functions and operators that are allowed for writing expressions in MySQL. Instructions for writing stored functions and user-defined functions are given in [Chapter 17, *Stored Procedures and Functions*](#), and [Section 24.2, “Adding New Functions to MySQL”](#).

An expression that contains NULL always produces a NULL value unless otherwise indicated in the documentation for a particular function or operator.

Note: By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. However, spaces around function arguments are permitted.

You can tell the MySQL server to accept spaces after function names by starting it with the `--sql-mode=IGNORE_SPACE` option. (See [Section 5.2.5, “The Server SQL Mode”](#).) Individual client programs can request this behavior by using the `CLIENT_IGNORE_SPACE` option for `mysql_real_connect()`. In either case, all function names become reserved words.

For the sake of brevity, most examples in this chapter display the output from the **mysql** program in abbreviated form. Rather than showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----+
| mod(29,9) |
+-----+
|          2 |
+-----+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
-> 2
```

12.1. Operators

12.1.1. Operator Precedence

Operator precedences are shown in the following list, from lowest precedence to the highest. Operators that are shown together on a line have the same precedence.

```
:=  
||, OR, XOR  
&&, AND  
NOT  
BETWEEN, CASE, WHEN, THEN, ELSE  
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN  
|  
&  
<<, >>  
-, +  
*, /, DIV, %, MOD  
^  
- (unary minus), ~ (unary bit inversion)  
!  
BINARY, COLLATE
```

The precedence shown for NOT is as of MySQL 5.0.2. For earlier versions, or from 5.0.2 on if the HIGH_NOT_PRECEDENCE SQL mode is enabled, the precedence of NOT is the same as that of the ! operator. See [Section 5.2.5, “The Server SQL Mode”](#).

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses. For example:

```
mysql> SELECT 1+2*3;  
-> 7  
mysql> SELECT (1+2)*3;  
-> 9
```

12.1.2. Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For

example, MySQL automatically converts numbers to strings as necessary, and vice versa.

```
mysql> SELECT 1+'1';  
      -> 2  
mysql> SELECT CONCAT(2, ' test');  
      -> '2 test'
```

It is also possible to perform explicit conversions. If you want to convert a number to a string explicitly, use the `CAST()` or `CONCAT()` function (`CAST()` is preferable):

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);  
      -> 38.8, '38.8'  
mysql> SELECT 38.8, CONCAT(38.8);  
      -> 38.8, '38.8'
```

The following rules describe how conversion occurs for comparison operations:

- If one or both arguments are `NULL`, the result of the comparison is `NULL`, except for the `NULL`-safe `<=>` equality comparison operator. For `NULL <=> NULL`, the result is true.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a `TIMESTAMP` or `DATETIME` column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. Note that this is not done for the arguments to `IN()`! To be safe, always use complete datetime, date, or time strings when doing comparisons.
- In all other cases, the arguments are compared as floating-point (real) numbers.

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
-> 0
mysql> SELECT 7 > '6x';
-> 1
mysql> SELECT 0 > 'x6';
-> 0
mysql> SELECT 0 = 'x6';
-> 1
```

Note that when you are comparing a string column with a number, MySQL cannot use an index on the column to look up the value quickly. If *str_col* is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value 1, such as '1', ' 1', or '1a'.

Comparisons that use floating-point numbers (or values that are converted to floating-point numbers) are approximate because such numbers are inexact. This might lead to results that appear inconsistent:

```
mysql> SELECT '18015376320243458' = 18015376320243458;
-> 1
mysql> SELECT '18015376320243459' = 18015376320243459;
-> 0
```

Such results can occur because the values are converted to floating-point numbers, which have only 53 bits of precision and are subject to rounding:

```
mysql> SELECT '18015376320243459'+0.0;
-> 1.8015376320243e+16
```

Furthermore, the conversion from string to floating-point and from integer to floating-point do not necessarily occur the same way. The integer may be converted to floating-point by the CPU, whereas the string is converted digit by digit in an operation that involves floating-point multiplications.

The results shown will vary on different systems, and can be affected by factors such as computer architecture or the compiler version or optimization level. One way to avoid such problems is to use `CAST()` so that a value will not be converted implicitly to a float-point number:

```
mysql> SELECT CAST('18015376320243459' AS UNSIGNED) = 18015376320243
-> 1
```

For more information about floating-point comparisons, see [Section A.5.8, “Problems with Floating-Point Comparisons”](#).

12.1.3. Comparison Functions and Operators

Comparison operations result in a value of 1 (TRUE), 0 (FALSE), or NULL. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

Some of the functions in this section (such as `LEAST()` and `GREATEST()`) return values other than 1 (TRUE), 0 (FALSE), or NULL. However, the value they return is based on comparison operations performed according to the rules described in [Section 12.1.2, “Type Conversion in Expression Evaluation”](#).

To convert a value to a specific type for comparison purposes, you can use the `CAST()` function. String values can be converted to a different character set using `CONVERT()`. See [Section 12.8, “Cast Functions and Operators”](#).

By default, string comparisons are not case sensitive and use the current character set. The default is `latin1` (cp1252 West European), which also works well for English.

- =

Equal:

```
mysql> SELECT 1 = 0;
-> 0
mysql> SELECT '0' = 0;
-> 1
mysql> SELECT '0.0' = 0;
-> 1
mysql> SELECT '0.01' = 0;
-> 0
mysql> SELECT '.01' = 0.01;
-> 1
```

- <=>

NULL-safe equal. This operator performs an equality comparison like the = operator, but returns 1 rather than NULL if both operands are NULL, and 0 rather than NULL if one operand is NULL.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
      -> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
      -> 1, NULL, NULL
```

- <>, !=

Not equal:

```
mysql> SELECT '.01' <> '0.01';
      -> 1
mysql> SELECT .01 <> '0.01';
      -> 0
mysql> SELECT 'zapp' <> 'zappp';
      -> 1
```

- <=

Less than or equal:

```
mysql> SELECT 0.1 <= 2;
      -> 1
```

- <

Less than:

```
mysql> SELECT 2 < 2;
      -> 0
```

- >=

Greater than or equal:

```
mysql> SELECT 2 >= 2;
      -> 1
```

- >

Greater than:

```
mysql> SELECT 2 > 2;
-> 0
```

- IS boolean_value, IS NOT boolean_value

Tests a value against a boolean value, where *boolean_value* can be TRUE, FALSE, or UNKNOWN.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
-> 1, 1, 1
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UN
-> 1, 1, 0
```

IS [NOT] boolean_value syntax was added in MySQL 5.0.2.

- IS NULL, IS NOT NULL

Tests whether a value is or is not NULL.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
-> 0, 0, 1
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
-> 1, 1, 0
```

To work well with ODBC programs, MySQL supports the following extra features when using IS NULL:

- You can find the row that contains the most recent AUTO_INCREMENT value by issuing a statement of the following form immediately after generating the value:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

This behavior can be disabled by setting SQL_AUTO_IS_NULL=0. See [Section 13.5.3, “SET Syntax”](#).

- For DATE and DATETIME columns that are declared as NOT NULL, you can find the special date '0000-00-00' by using a statement like this:

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

This is needed to get some ODBC applications to work because ODBC does not support a '0000-00-00' date value.

- `expr BETWEEN min AND max`

If `expr` is greater than or equal to `min` and `expr` is less than or equal to `max`, `BETWEEN` returns 1, otherwise it returns 0. This is equivalent to the expression `(min <= expr AND expr <= max)` if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described in [Section 12.1.2, “Type Conversion in Expression Evaluation”](#), but applied to all the three arguments.

```
mysql> SELECT 1 BETWEEN 2 AND 3;
      -> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
      -> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
      -> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
      -> 0
```

- `expr NOT BETWEEN min AND max`

This is the same as `NOT (expr BETWEEN min AND max)`.

- `COALESCE(value,...)`

Returns the first non-NULL value in the list, or NULL if there are no non-NULL values.

```
mysql> SELECT COALESCE(NULL, 1);
      -> 1
mysql> SELECT COALESCE(NULL, NULL, NULL);
      -> NULL
```

- `GREATEST(value1,value2,...)`

With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for `LEAST()`.

```
mysql> SELECT GREATEST(2, 0);
      -> 2
mysql> SELECT GREATEST(34.0, 3.0, 5.0, 767.0);
      -> 767.0
mysql> SELECT GREATEST('B', 'A', 'C');
      -> 'C'
```

Before MySQL 5.0.13, `GREATEST()` returns `NULL` only if all arguments are `NULL`. As of 5.0.13, it returns `NULL` if any argument is `NULL`.

- `expr IN (value,...)`

Returns 1 if `expr` is equal to any of the values in the `IN` list, else returns 0. If all values are constants, they are evaluated according to the type of `expr` and sorted. The search for the item then is done using a binary search. This means `IN` is very quick if the `IN` value list consists entirely of constants. Otherwise, type conversion takes place according to the rules described in [Section 12.1.2, “Type Conversion in Expression Evaluation”](#), but applied to all the arguments.

```
mysql> SELECT 2 IN (0,3,5,7);
-> 0
mysql> SELECT 'wefwf' IN ('wee', 'wefwf', 'weg');
-> 1
```

You should never mix quoted and unquoted values in an `IN` list because the comparison rules for quoted values (such as strings) and unquoted values (such as numbers) differ. Mixing types may therefore lead to inconsistent results. For example, do not write an `IN` expression like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2, 'a');
```

Instead, write it like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1', '2', 'a');
```

The number of values in the `IN` list is only limited by the `max_allowed_packet` value.

To comply with the SQL standard, `IN` returns `NULL` not only if the expression on the left hand side is `NULL`, but also if no match is found in the list and one of the expressions in the list is `NULL`.

`IN()` syntax can also be used to write certain types of subqueries. See [Section 13.2.8.3, “Subqueries with ANY, IN, and SOME”](#).

- `expr NOT IN (value,...)`

This is the same as `NOT (expr IN (value,...))`.

- `ISNULL(expr)`

If *expr* is `NULL`, `ISNULL()` returns 1, otherwise it returns 0.

```
mysql> SELECT ISNULL(1+1);  
      -> 0  
mysql> SELECT ISNULL(1/0);  
      -> 1
```

`ISNULL()` can be used instead of `=` to test whether a value is `NULL`. (Comparing a value to `NULL` using `=` always yields false.)

The `ISNULL()` function shares some special behaviors with the `IS NULL` comparison operator. See the description of `IS NULL`.

- `INTERVAL(N,N1,N2,N3,...)`

Returns 0 if $N < N1$, 1 if $N < N2$ and so on or -1 if N is `NULL`. All arguments are treated as integers. It is required that $N1 < N2 < N3 < \dots < Nn$ for this function to work correctly. This is because a binary search is used (very fast).

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);  
      -> 3  
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);  
      -> 2  
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);  
      -> 0
```

- `LEAST(value1,value2,...)`

With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

- If the return value is used in an `INTEGER` context or all arguments are integer-valued, they are compared as integers.
- If the return value is used in a `REAL` context or all arguments are real-valued, they are compared as reals.
- If any argument is a case-sensitive string, the arguments are compared as case-sensitive strings.

- In all other cases, the arguments are compared as case-insensitive strings.

Before MySQL 5.0.13, `LEAST()` returns `NULL` only if all arguments are `NULL`. As of 5.0.13, it returns `NULL` if any argument is `NULL`.

```
mysql> SELECT LEAST(2, 0);
      -> 0
mysql> SELECT LEAST(34.0, 3.0, 5.0, 767.0);
      -> 3.0
mysql> SELECT LEAST('B', 'A', 'C');
      -> 'A'
```

Note that the preceding conversion rules can produce strange results in some borderline cases:

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED)
      -> -9223372036854775808
```

This happens because MySQL reads `9223372036854775808.0` in an integer context. The integer representation is not good enough to hold the value, so it wraps to a signed integer.

12.1.4. Logical Operators

In SQL, all logical operators evaluate to `TRUE`, `FALSE`, or `NULL` (`UNKNOWN`). In MySQL, these are implemented as `1` (`TRUE`), `0` (`FALSE`), and `NULL`. Most of this is common to different SQL database servers, although some servers may return any non-zero value for `TRUE`.

- `NOT`, `!`

Logical `NOT`. Evaluates to `1` if the operand is `0`, to `0` if the operand is non-zero, and `NOT NULL` returns `NULL`.

```
mysql> SELECT NOT 10;
      -> 0
mysql> SELECT NOT 0;
      -> 1
mysql> SELECT NOT NULL;
      -> NULL
mysql> SELECT ! (1+1);
```

```
mysql> SELECT ! 1+1;
      -> 0
      -> 1
```

The last example produces 1 because the expression evaluates the same way as (!1)+1.

Note that the precedence of the NOT operator changed in MySQL 5.0.2. See [Section 12.1.1, “Operator Precedence”](#).

- AND, &&

Logical AND. Evaluates to 1 if all operands are non-zero and not NULL, to 0 if one or more operands are 0, otherwise NULL is returned.

```
mysql> SELECT 1 && 1;
      -> 1
mysql> SELECT 1 && 0;
      -> 0
mysql> SELECT 1 && NULL;
      -> NULL
mysql> SELECT 0 && NULL;
      -> 0
mysql> SELECT NULL && 0;
      -> 0
```

- OR, ||

Logical OR. When both operands are non-NULL, the result is 1 if any operand is non-zero, and 0 otherwise. With a NULL operand, the result is 1 if the other operand is non-zero, and NULL otherwise. If both operands are NULL, the result is NULL.

```
mysql> SELECT 1 || 1;
      -> 1
mysql> SELECT 1 || 0;
      -> 1
mysql> SELECT 0 || 0;
      -> 0
mysql> SELECT 0 || NULL;
      -> NULL
mysql> SELECT 1 || NULL;
      -> 1
```

- XOR

Logical XOR. Returns NULL if either operand is NULL. For non-NULL operands, evaluates to 1 if an odd number of operands is non-zero, otherwise 0 is returned.

```
mysql> SELECT 1 XOR 1;  
      -> 0  
mysql> SELECT 1 XOR 0;  
      -> 1  
mysql> SELECT 1 XOR NULL;  
      -> NULL  
mysql> SELECT 1 XOR 1 XOR 1;  
      -> 1
```

$a \text{ XOR } b$ is mathematically equal to $(a \text{ AND } (\text{NOT } b)) \text{ OR } ((\text{NOT } a) \text{ AND } b)$.

12.2. Control Flow Functions

- CASE *value* WHEN [*compare_value*] THEN *result* [WHEN [*compare_value*] THEN *result* ...] [ELSE *result*] END

```
CASE WHEN [condition] THEN result [WHEN [condition] THEN
result ...] [ELSE result] END
```

The first version returns the *result* where *value*=*compare_value*. The second version returns the result for the first condition that is true. If there was no matching result value, the result after ELSE is returned, or NULL if there is no ELSE part.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
->      WHEN 2 THEN 'two' ELSE 'more' END;
->      'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
->      'true'
mysql> SELECT CASE BINARY 'B'
->      WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
->      NULL
```

The default return type of a CASE expression is the compatible aggregated type of all return values, but also depends on the context in which it is used. If used in a string context, the result is returned as a string. If used in a numeric context, then the result is returned as a decimal, real, or integer value.

Note: The syntax of the CASE *expression* shown here differs slightly from that of the SQL CASE *statement* described in [Section 17.2.10.2, “CASE Statement”](#), for use inside stored routines. The CASE statement cannot have an ELSE NULL clause, and it is terminated with END CASE instead of END.

- IF(*expr1*,*expr2*,*expr3*)

If *expr1* is TRUE (*expr1* <> 0 and *expr1* <> NULL) then IF() returns *expr2*; otherwise it returns *expr3*. IF() returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IF(1>2, 2, 3);
```

```

mysql> SELECT IF(1<2, 'yes', 'no');
      -> 'yes'
mysql> SELECT IF(STRCMP('test', 'test1'), 'no', 'yes');
      -> 'no'

```

If only one of *expr2* or *expr3* is explicitly NULL, the result type of the IF() function is the type of the non-NULL expression.

expr1 is evaluated as an integer value, which means that if you are testing floating-point or string values, you should do so using a comparison operation.

```

mysql> SELECT IF(0.1, 1, 0);
      -> 0
mysql> SELECT IF(0.1<>0, 1, 0);
      -> 1

```

In the first case shown, IF(0.1) returns 0 because 0.1 is converted to an integer value, resulting in a test of IF(0). This may not be what you expect. In the second case, the comparison tests the original floating-point value to see whether it is non-zero. The result of the comparison is used as an integer.

The default return type of IF() (which may matter when it is stored into a temporary table) is calculated as follows:

Expression	Return Value
<i>expr2</i> or <i>expr3</i> returns a string	string
<i>expr2</i> or <i>expr3</i> returns a floating-point value	floating-point
<i>expr2</i> or <i>expr3</i> returns an integer	integer

If *expr2* and *expr3* are both strings, the result is case sensitive if either string is case sensitive.

Note: There is also an IF *statement*, which differs from the IF() *function* described here. See [Section 17.2.10.1, “IF Statement”](#).

- IFNULL(*expr1*,*expr2*)

If *expr1* is not NULL, IFNULL() returns *expr1*; otherwise it returns *expr2*.

IFNULL() returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IFNULL(1,0);
      -> 1
mysql> SELECT IFNULL(NULL,10);
      -> 10
mysql> SELECT IFNULL(1/0,10);
      -> 10
mysql> SELECT IFNULL(1/0,'yes');
      -> 'yes'
```

The default result value of IFNULL(*expr1*,*expr2*) is the more “general” of the two expressions, in the order STRING, REAL, or INTEGER. Consider the case of a table based on expressions or where MySQL must internally store a value returned by IFNULL() in a temporary table:

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| test  | char(4)|      |     |          |       |
+-----+-----+-----+-----+-----+-----+
```

In this example, the type of the test column is CHAR(4).

- NULLIF(*expr1*,*expr2*)

Returns NULL if *expr1* = *expr2* is true, otherwise returns *expr1*. This is the same as CASE WHEN *expr1* = *expr2* THEN NULL ELSE *expr1* END.

```
mysql> SELECT NULLIF(1,1);
      -> NULL
mysql> SELECT NULLIF(1,2);
      -> 1
```

Note that MySQL evaluates *expr1* twice if the arguments are not equal.

12.3. String Functions

String-valued functions return NULL if the length of the result would be greater than the value of the `max_allowed_packet` system variable. See [Section 7.5.2, “Tuning Server Parameters”](#).

For functions that operate on string positions, the first position is numbered 1.

- `ASCII(str)`

Returns the numeric value of the leftmost character of the string *str*. Returns 0 if *str* is the empty string. Returns NULL if *str* is NULL. `ASCII()` works for characters with numeric values from 0 to 255.

```
mysql> SELECT ASCII('2');
      -> 50
mysql> SELECT ASCII(2);
      -> 50
mysql> SELECT ASCII('dx');
      -> 100
```

See also the `ORD()` function.

- `BIN(N)`

Returns a string representation of the binary value of *N*, where *N* is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,2)`. Returns NULL if *N* is NULL.

```
mysql> SELECT BIN(12);
      -> '1100'
```

- `BIT_LENGTH(str)`

Returns the length of the string *str* in bits.

```
mysql> SELECT BIT_LENGTH('text');
      -> 32
```

- `CHAR(N,... [USING charset_name])`

CHAR() interprets each argument *N* as an integer and returns a string consisting of the characters given by the code values of those integers. NULL values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
      -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
      -> 'MMM'
```

As of MySQL 5.0.15, CHAR() arguments larger than 255 are converted into multiple result bytes. For example, CHAR(256) is equivalent to CHAR(1,0), and CHAR(256*256) is equivalent to CHAR(1,0,0):

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+-----+-----+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+-----+-----+
| 0100          | 0100          |
+-----+-----+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+-----+-----+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+-----+-----+
| 010000         | 010000         |
+-----+-----+
```

By default, CHAR() returns a binary string. To produce a string in a given character set, use the optional USING clause:

```
mysql> SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8))
+-----+-----+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+-----+-----+
| binary              | utf8                          |
+-----+-----+
```

If USING is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from CHAR() becomes NULL.

Before MySQL 5.0.15, CHAR() returns a string in the connection character set and the USING clause is unavailable. In addition, each argument is interpreted modulo 256, so CHAR(256) and CHAR(256*256) both are equivalent to CHAR(0).

- `CHAR_LENGTH(str)`

Returns the length of the string *str*, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, `LENGTH()` returns 10, whereas `CHAR_LENGTH()` returns 5.

- `CHARACTER_LENGTH(str)`

`CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

- `CONCAT(str1,str2,...)`

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:

```
SELECT CONCAT(CAST(int_col AS CHAR), char_col);
```

`CONCAT()` returns `NULL` if any argument is `NULL`.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
      -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
      -> NULL
mysql> SELECT CONCAT(14.3);
      -> '14.3'
```

- `CONCAT_WS(separator,str1,str2,...)`

`CONCAT_WS()` stands for Concatenate With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

```
mysql> SELECT CONCAT_WS(',', 'First name', 'Second name', 'Last Name');
      -> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(',', 'First name', NULL, 'Last Name');
```

-> 'First name,Last Name'

CONCAT_WS() does not skip empty strings. However, it does skip any NULL values after the separator argument.

- CONV(*N*,*from_base*,*to_base*)

Converts numbers between different number bases. Returns a string representation of the number *N*, converted from base *from_base* to base *to_base*. Returns NULL if any argument is NULL. The argument *N* is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If *to_base* is a negative number, *N* is regarded as a signed number. Otherwise, *N* is treated as unsigned. CONV() works with 64-bit precision.

```
mysql> SELECT CONV('a',16,2);
      -> '1010'
mysql> SELECT CONV('6E',18,8);
      -> '172'
mysql> SELECT CONV(-17,10,-18);
      -> '-H'
mysql> SELECT CONV(10+'10'+ '10'+0xa,10,10);
      -> '40'
```

- ELT(*N*,*str1*,*str2*,*str3*,...)

Returns *str1* if *N* = 1, *str2* if *N* = 2, and so on. Returns NULL if *N* is less than 1 or greater than the number of arguments. ELT() is the complement of FIELD().

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
      -> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
      -> 'foo'
```

- EXPORT_SET(*bits*,*on*,*off*[,*separator*[,*number_of_bits*]])

Returns a string such that for every bit set in the value *bits*, you get an *on* string and for every reset bit, you get an *off* string. Bits in *bits* are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the *separator* string (the default being the comma character ','). The number of bits examined is given by *number_of_bits* (defaults to 64).

```
mysql> SELECT EXPORT_SET(5, 'Y', 'N', ' ', ' ', 4);
      -> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6, '1', '0', ' ', ' ', 10);
      -> '0,1,1,0,0,0,0,0,0,0'
```

- FIELD(*str*,*str1*,*str2*,*str3*,...)

Returns the index (position) of *str* in the *str1*, *str2*, *str3*, . . . list. Returns 0 if *str* is not found.

If all arguments to FIELD() are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

If *str* is NULL, the return value is 0 because NULL fails equality comparison with any value. FIELD() is the complement of ELT().

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
      -> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
      -> 0
```

- FIND_IN_SET(*str*,*strlist*)

Returns a value in the range of 1 to *N* if the string *str* is in the string list *strlist* consisting of *N* substrings. A string list is a string composed of substrings separated by ',' characters. If the first argument is a constant string and the second is a column of type SET, the FIND_IN_SET() function is optimized to use bit arithmetic. Returns 0 if *str* is not in *strlist* or if *strlist* is the empty string. Returns NULL if either argument is NULL. This function does not work properly if the first argument contains a comma (',') character.

```
mysql> SELECT FIND_IN_SET('b', 'a,b,c,d');
      -> 2
```

- FORMAT(*X*,*D*)

Formats the number *X* to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string. If *D* is 0, the result has no decimal point or fractional part.

```
mysql> SELECT FORMAT(12332.123456, 4);
      -> '12,332.1235'
mysql> SELECT FORMAT(12332.1, 4);
      -> '12,332.1000'
mysql> SELECT FORMAT(12332.2, 0);
      -> '12,332'
```

- HEX(*N_or_S*)

If *N_or_S* is a number, returns a string representation of the hexadecimal value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to CONV(*N*,10,16).

If *N_or_S* is a string, returns a hexadecimal string representation of *N_or_S* where each character in *N_or_S* is converted to two hexadecimal digits.

```
mysql> SELECT HEX(255);
      -> 'FF'
mysql> SELECT 0x616263;
      -> 'abc'
mysql> SELECT HEX('abc');
      -> 616263
```

- INSERT(*str,pos,len,newstr*)

Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string *newstr*. Returns the original string if *pos* is not within the length of the string. Replaces the rest of the string from position *pos* if *len* is not within the length of the rest of the string. Returns NULL if any argument is NULL.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
      -> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
      -> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
      -> 'QuWhat'
```

This function is multi-byte safe.

- INSTR(*str,substr*)

Returns the position of the first occurrence of substring *substr* in string *str*. This is the same as the two-argument form of LOCATE(), except that the

order of the arguments is reversed.

```
mysql> SELECT INSTR('foobarbar', 'bar');
      -> 4
mysql> SELECT INSTR('xbar', 'foobar');
      -> 0
```

This function is multi-byte safe, and is case sensitive only if at least one argument is a binary string.

- `LCASE(str)`

`LCASE()` is a synonym for `LOWER()`.

- `LEFT(str,len)`

Returns the leftmost *len* characters from the string *str*, or `NULL` if any argument is `NULL`.

```
mysql> SELECT LEFT('foobarbar', 5);
      -> 'fooba'
```

- `LENGTH(str)`

Returns the length of the string *str*, measured in bytes. A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `LENGTH()` returns 10, whereas `CHAR_LENGTH()` returns 5.

```
mysql> SELECT LENGTH('text');
      -> 4
```

- `LOAD_FILE(file_name)`

Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full pathname to the file, and you must have the `FILE` privilege. The file must be readable by all and its size less than `max_allowed_packet` bytes.

If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

As of MySQL 5.0.19, the `character_set_filesystem` system variable controls interpretation of filenames that are given as literal strings.

```
mysql> UPDATE t
        SET blob_col=LOAD_FILE('/tmp/picture')
        WHERE id=1;
```

- `LOCATE(substr,str)`, `LOCATE(substr,str,pos)`

The first syntax returns the position of the first occurrence of substring *substr* in string *str*. The second syntax returns the position of the first occurrence of substring *substr* in string *str*, starting at position *pos*. Returns 0 if *substr* is not in *str*.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
        -> 7
```

This function is multi-byte safe, and is case-sensitive only if at least one argument is a binary string.

- `LOWER(str)`

Returns the string *str* with all characters changed to lowercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

This function is multi-byte safe.

- `LPAD(str,len,padstr)`

Returns the string *str*, left-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.

```
mysql> SELECT LPAD('hi',4,'??');
        -> '??hi'
```

```
mysql> SELECT LPAD('hi',1,'??');
      -> 'h'
```

- LTRIM(str)

Returns the string *str* with leading space characters removed.

```
mysql> SELECT LTRIM('  barbar');
      -> 'barbar'
```

This function is multi-byte safe.

- MAKE_SET(bits,str1,str2,...)

Returns a set value (a string containing substrings separated by ',' characters) consisting of the strings that have the corresponding bit in *bits* set. *str1* corresponds to bit 0, *str2* to bit 1, and so on. NULL values in *str1*, *str2*, ... are not appended to the result.

```
mysql> SELECT MAKE_SET(1, 'a', 'b', 'c');
      -> 'a'
mysql> SELECT MAKE_SET(1 | 4, 'hello', 'nice', 'world');
      -> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4, 'hello', 'nice', NULL, 'world');
      -> 'hello'
mysql> SELECT MAKE_SET(0, 'a', 'b', 'c');
      -> ''
```

- MID(str,pos,len)

MID(str,pos,len) is a synonym for SUBSTRING(str,pos,len).

- OCT(N)

Returns a string representation of the octal value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to CONV(N,10,8). Returns NULL if *N* is NULL.

```
mysql> SELECT OCT(12);
      -> '14'
```

- OCTET_LENGTH(str)

OCTET_LENGTH() is a synonym for LENGTH().

- `ORD(str)`

If the leftmost character of the string *str* is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
(1st byte code)
+ (2nd byte code × 256)
+ (3rd byte code × 2562) ...
```

If the leftmost character is not a multi-byte character, `ORD()` returns the same value as the `ASCII()` function.

```
mysql> SELECT ORD('2');
-> 50
```

- `POSITION(substr IN str)`

`POSITION(substr IN str)` is a synonym for `LOCATE(substr, str)`.

- `QUOTE(str)`

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotes and with each instance of single quote (''), backslash ('\'), ASCII NUL, and Control-Z preceded by a backslash. If the argument is `NULL`, the return value is the word "NULL" without enclosing single quotes.

```
mysql> SELECT QUOTE('Don\'t!');
-> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
-> NULL
```

- `REPEAT(str, count)`

Returns a string consisting of the string *str* repeated *count* times. If *count* is less than 1, returns an empty string. Returns `NULL` if *str* or *count* are `NULL`.

```
mysql> SELECT REPEAT('MySQL', 3);
-> 'MySQLMySQLMySQL'
```

- `REPLACE(str,from_str,to_str)`

Returns the string *str* with all occurrences of the string *from_str* replaced by the string *to_str*. `REPLACE()` performs a case-sensitive match when searching for *from_str*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
      -> 'WwWwWw.mysql.com'
```

This function is multi-byte safe.

- `REVERSE(str)`

Returns the string *str* with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
      -> 'cba'
```

This function is multi-byte safe.

- `RIGHT(str,len)`

Returns the rightmost *len* characters from the string *str*, or NULL if any argument is NULL.

```
mysql> SELECT RIGHT('foobarbar', 4);
      -> 'rbar'
```

This function is multi-byte safe.

- `RPAD(str,len,padstr)`

Returns the string *str*, right-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.

```
mysql> SELECT RPAD('hi',5,'?');
      -> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
      -> 'h'
```

This function is multi-byte safe.

- RTRIM(*str*)

Returns the string *str* with trailing space characters removed.

```
mysql> SELECT RTRIM('barbar  ');
        -> 'barbar'
```

This function is multi-byte safe.

- SOUNDEX(*str*)

Returns a soundex string from *str*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the SOUNDEX() function returns an arbitrarily long string. You can use SUBSTRING() on the result to get a standard soundex string. All non-alphabetic characters in *str* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

```
mysql> SELECT SOUNDEX('Hello');
        -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
        -> 'Q36324'
```

Note: This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- *expr1* SOUNDS LIKE *expr2*

This is the same as SOUNDEX(*expr1*) = SOUNDEX(*expr2*).

- SPACE(*N*)

Returns a string consisting of *N* space characters.

```
mysql> SELECT SPACE(6);
        -> '      '
```

- SUBSTRING(*str, pos*), SUBSTRING(*str* FROM *pos*),
SUBSTRING(*str, pos, len*), SUBSTRING(*str* FROM *pos* FOR *len*)

The forms without a *len* argument return a substring from string *str* starting at position *pos*. The forms with a *len* argument return a substring *len* characters long from string *str*, starting at position *pos*. The forms that use FROM are standard SQL syntax. It is also possible to use a negative value for *pos*. In this case, the beginning of the substring is *pos* characters from the end of the string, rather than the beginning. A negative value may be used for *pos* in any of the forms of this function.

```
mysql> SELECT SUBSTRING('Quadratically',5);
      -> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
      -> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
      -> 'ratica'
mysql> SELECT SUBSTRING('Sakila', -3);
      -> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
      -> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
      -> 'ki'
```

This function is multi-byte safe.

If *len* is less than 1, the result is the empty string.

SUBSTR() is a synonym for SUBSTRING().

- SUBSTRING_INDEX(*str*,*delim*,*count*)

Returns the substring from string *str* before *count* occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned. SUBSTRING_INDEX() performs a case-sensitive match when searching for *delim*.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
      -> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
      -> 'mysql.com'
```

This function is multi-byte safe.

- `TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str),`
`TRIM([remstr FROM] str)`

Returns the string *str* with all *remstr* prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. *remstr* is optional and, if not specified, spaces are removed.

```
mysql> SELECT TRIM('  bar  ');
      -> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
      -> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
      -> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');
      -> 'barx'
```

This function is multi-byte safe.

- `UCASE(str)`

`UCASE()` is a synonym for `UPPER()`.

- `UNHEX(str)`

Performs the inverse operation of `HEX(str)`. That is, it interprets each pair of hexadecimal digits in the argument as a number and converts it to the character represented by the number. The resulting characters are returned as a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
      -> 'MySQL'
mysql> SELECT 0x4D7953514C;
      -> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
      -> 'string'
mysql> SELECT HEX(UNHEX('1267'));
      -> '1267'
```

- `UPPER(str)`

Returns the string *str* with all characters changed to uppercase according to the current character set mapping. The default is `latin1 (cp1252 West European)`.

```
mysql> SELECT UPPER('Hej');
      -> 'HEJ'
```

This function is multi-byte safe.

12.3.1. String Comparison Functions

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

- `expr LIKE pat [ESCAPE 'escape_char']`

Pattern matching using SQL simple regular expression comparison. Returns 1 (TRUE) or 0 (FALSE). If either *expr* or *pat* is NULL, the result is NULL.

The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

Per the SQL standard, LIKE performs matching on a per-character basis, thus it can produce results different from the = comparison operator:

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----+
|                                          0 |
+-----+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-----+
|                                          1 |
+-----+
```

With LIKE you can use the following two wildcard characters in the pattern:

Character	Description
%	Matches any number of characters, even zero characters

||— ||Matches exactly one character ||

```
mysql> SELECT 'David!' LIKE 'David_';
-> 1
mysql> SELECT 'David!' LIKE '%D%v%';
-> 1
```

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the ESCAPE character, ‘\’ is assumed.

String	Description
\%	Matches one ‘%’ character
_	Matches one ‘_’ character

```
mysql> SELECT 'David!' LIKE 'David\_';
-> 0
mysql> SELECT 'David_' LIKE 'David\_';
-> 1
```

To specify a different escape character, use the ESCAPE clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
-> 1
```

The escape sequence should be empty or one character long. As of MySQL 5.0.16, if the NO_BACKSLASH_ESCAPES SQL mode is enabled, the sequence cannot be empty.

The following two statements illustrate that string comparisons are not case sensitive unless one of the operands is a binary string:

```
mysql> SELECT 'abc' LIKE 'ABC';
-> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
-> 0
```

In MySQL, LIKE is allowed on numeric expressions. (This is an extension to the standard SQL LIKE.)

```
mysql> SELECT 10 LIKE '1%';
-> 1
```

Note: Because MySQL uses C escape syntax in strings (for example, ‘\n’ to

represent a newline character), you must double any ‘\’ that you use in LIKE strings. For example, to search for ‘\n’, specify it as ‘\\n’. To search for ‘\’, specify it as ‘\\\\’; this is because the backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against. (Exception: At the end of the pattern string, backslash can be specified as ‘\’. At the end of the string, backslash stands for itself because there is nothing following to escape.)

- `expr NOT LIKE pat [ESCAPE 'escape_char']`

This is the same as `NOT (expr LIKE pat [ESCAPE 'escape_char'])`.

- `expr NOT REGEXP pat, expr NOT RLIKE pat`

This is the same as `NOT (expr REGEXP pat)`.

- `expr REGEXP pat expr RLIKE pat`

Performs a pattern match of a string expression *expr* against a pattern *pat*. The pattern can be an extended regular expression. The syntax for regular expressions is discussed in [Appendix G, Regular Expressions](#). Returns 1 if *expr* matches *pat*; otherwise it returns 0. If either *expr* or *pat* is NULL, the result is NULL. RLIKE is a synonym for REGEXP, provided for MySQL compatibility.

The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

Note: Because MySQL uses the C escape syntax in strings (for example, ‘\n’ to represent the newline character), you must double any ‘\’ that you use in your REGEXP strings.

REGEXP is not case sensitive, except when used with binary strings.

```
mysql> SELECT 'Monty!' REGEXP 'm%y%';
-> 0
mysql> SELECT 'Monty!' REGEXP '.*';
-> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
-> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
-> 1 0
```

```
mysql> SELECT 'a' REGEXP '^[a-d]';  
-> 1
```

REGEXP and RLIKE use the current character set when deciding the type of a character. The default is `latin1 (cp1252 West European)`. **Warning:** These operators are not multi-byte safe.

- `STRCMP(expr1,expr2)`

`STRCMP()` returns 0 if the strings are the same, -1 if the first argument is smaller than the second according to the current sort order, and 1 otherwise.

```
mysql> SELECT STRCMP('text', 'text2');  
-> -1  
mysql> SELECT STRCMP('text2', 'text');  
-> 1  
mysql> SELECT STRCMP('text', 'text');  
-> 0
```

`STRCMP()` uses the current character set when performing comparisons. This makes the default comparison behavior case insensitive unless one or both of the operands are binary strings.

12.4. Numeric Functions

12.4.1. Arithmetic Operators

The usual arithmetic operators are available. The precision of the result is determined according to the following rules:

- Note that in the case of -, +, and *, the result is calculated with BIGINT (64-bit) precision if both arguments are integers.
- If one of the arguments is an unsigned integer, and the other argument is also an integer, the result is an unsigned integer.
- If any of the operands of a +, -, /, *, % is a real or string value, then the precision of the result is the precision of the argument with the maximum precision.
- In multiplication and division, the precision of the result when using two integer values is the precision of the first argument + the value of the `div_precision_increment` global variable. For example, the expression `5.05 / 0.0014` would have a precision of six decimal places (`4.047976`).

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, `(14620 / 9432456) / (24250 / 9432456)`, would resolve first to `(0.0014) / (0.0026)`, with the final result having 8 decimal places (`0.57692308`).

Because of these rules and the method they are applied, care should be taken to ensure that components and sub-components of a calculation use the appropriate level of precision. See [Section 12.8, “Cast Functions and Operators”](#).

- +

Addition:

```
mysql> SELECT 3+5;  
-> 8
```

- -

Subtraction:

```
mysql> SELECT 3-5;  
-> -2
```

- -

Unary minus. This operator changes the sign of the argument.

```
mysql> SELECT - 2;  
-> -2
```

Note: If this operator is used with a `BIGINT`, the return value is also a `BIGINT`. This means that you should avoid using `-` on integers that may have the value of -2^{63} .

- *

Multiplication:

```
mysql> SELECT 3*5;  
-> 15  
mysql> SELECT 18014398509481984*18014398509481984.0;  
-> 324518553658426726783156020576256.0  
mysql> SELECT 18014398509481984*18014398509481984;  
-> 0
```

The result of the last expression is incorrect because the result of the integer multiplication exceeds the 64-bit range of `BIGINT` calculations. (See [Section 11.2, “Numeric Types”](#).)

- /

Division:

```
mysql> SELECT 3/5;  
-> 0.60
```

Division by zero produces a `NULL` result:

```
mysql> SELECT 102/(1-1);  
-> NULL
```

A division is calculated with `BIGINT` arithmetic only if performed in a

context where its result is converted to an integer.

- DIV

Integer division. Similar to `FLOOR()`, but is safe with `BIGINT` values.

```
mysql> SELECT 5 DIV 2;  
-> 2
```

12.4.2. Mathematical Functions

All mathematical functions return `NULL` in the event of an error.

- `ABS(X)`

Returns the absolute value of x .

```
mysql> SELECT ABS(2);  
-> 2  
mysql> SELECT ABS(-32);  
-> 32
```

This function is safe to use with `BIGINT` values.

- `ACOS(X)`

Returns the arc cosine of x , that is, the value whose cosine is x . Returns `NULL` if x is not in the range -1 to 1 .

```
mysql> SELECT ACOS(1);  
-> 0  
mysql> SELECT ACOS(1.0001);  
-> NULL  
mysql> SELECT ACOS(0);  
-> 1.5707963267949
```

- `ASIN(X)`

Returns the arc sine of x , that is, the value whose sine is x . Returns `NULL` if x is not in the range -1 to 1 .

```
mysql> SELECT ASIN(0.2);  
-> 0.20135792079033
```

```
mysql> SELECT ASIN('foo');

+-----+
| ASIN('foo') |
+-----+
|           0 |
+-----+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS;
+-----+-----+-----+-----+
| Level   | Code | Message                                     |
+-----+-----+-----+-----+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+-----+-----+-----+-----+
```

- ATAN(X)

Returns the arc tangent of x , that is, the value whose tangent is x .

```
mysql> SELECT ATAN(2);
-> 1.1071487177941
mysql> SELECT ATAN(-2);
-> -1.1071487177941
```

- ATAN(Y,X), ATAN2(Y,X)

Returns the arc tangent of the two variables x and y . It is similar to calculating the arc tangent of y / x , except that the signs of both arguments are used to determine the quadrant of the result.

```
mysql> SELECT ATAN(-2, 2);
-> -0.78539816339745
mysql> SELECT ATAN2(PI(), 0);
-> 1.5707963267949
```

- CEILING(X), CEIL(X)

Returns the smallest integer value not less than x .

```
mysql> SELECT CEILING(1.23);
-> 2
mysql> SELECT CEIL(-1.23);
-> -1
```

These two functions are synonymous. Note that the return value is

converted to a BIGINT.

- **COS(X)**

Returns the cosine of x , where x is given in radians.

```
mysql> SELECT COS(PI());  
-> -1
```

- **COT(X)**

Returns the cotangent of x .

```
mysql> SELECT COT(12);  
-> -1.5726734063977  
mysql> SELECT COT(0);  
-> NULL
```

- **CRC32(expr)**

Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is NULL if the argument is NULL. The argument is expected to be a string and (if possible) is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');  
-> 3259397556  
mysql> SELECT CRC32('mysql');  
-> 2501908538
```

- **DEGREES(X)**

Returns the argument x , converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());  
-> 180  
mysql> SELECT DEGREES(PI() / 2);  
-> 90
```

- **EXP(X)**

Returns the value of e (the base of natural logarithms) raised to the power of x .

```
mysql> SELECT EXP(2);
```

```
mysql> SELECT EXP(-2);
-> 7.3890560989307
mysql> SELECT EXP(0);
-> 0.13533528323661
mysql> SELECT EXP(0);
-> 1
```

- FLOOR(X)

Returns the largest integer value not greater than x .

```
mysql> SELECT FLOOR(1.23);
-> 1
mysql> SELECT FLOOR(-1.23);
-> -2
```

Note that the return value is converted to a BIGINT.

- FORMAT(X,D)

Formats the number x to a format like '#,###,###.##', rounded to D decimal places, and returns the result as a string. For details, see [Section 12.3, “String Functions”](#).

- LN(X)

Returns the natural logarithm of x ; that is, the base- e logarithm of x .

```
mysql> SELECT LN(2);
-> 0.69314718055995
mysql> SELECT LN(-2);
-> NULL
```

This function is synonymous with LOG(X).

- LOG(X), LOG(B,X)

If called with one parameter, this function returns the natural logarithm of x .

```
mysql> SELECT LOG(2);
-> 0.69314718055995
mysql> SELECT LOG(-2);
-> NULL
```

If called with two parameters, this function returns the logarithm of x for an

arbitrary base B .

```
mysql> SELECT LOG(2, 65536);  
      -> 16  
mysql> SELECT LOG(10, 100);  
      -> 2
```

$\text{LOG}(B, X)$ is equivalent to $\text{LOG}(X) / \text{LOG}(B)$.

- $\text{LOG}_2(X)$

Returns the base-2 logarithm of x .

```
mysql> SELECT LOG2(65536);  
      -> 16  
mysql> SELECT LOG2(-100);  
      -> NULL
```

$\text{LOG}_2()$ is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression $\text{LOG}(X) / \text{LOG}(2)$.

- $\text{LOG}_{10}(X)$

Returns the base-10 logarithm of x .

```
mysql> SELECT LOG10(2);  
      -> 0.30102999566398  
mysql> SELECT LOG10(100);  
      -> 2  
mysql> SELECT LOG10(-100);  
      -> NULL
```

$\text{LOG}_{10}(X)$ is equivalent to $\text{LOG}(10, X)$.

- $\text{MOD}(N, M)$, $N \% M$, $N \text{ MOD } M$

Modulo operation. Returns the remainder of N divided by M .

```
mysql> SELECT MOD(234, 10);  
      -> 4  
mysql> SELECT 253 % 7;  
      -> 1  
mysql> SELECT MOD(29, 9);  
      -> 2  
mysql> SELECT 29 MOD 9;
```

-> 2

This function is safe to use with BIGINT values.

MOD() also works on values that have a fractional part and returns the exact remainder after division:

```
mysql> SELECT MOD(34.5, 3);  
      -> 1.5
```

- PI()

Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

```
mysql> SELECT PI();  
      -> 3.141593  
mysql> SELECT PI()+0.000000000000000000;  
      -> 3.141592653589793116
```

- POW(X,Y), POWER(X,Y)

Returns the value of x raised to the power of y .

```
mysql> SELECT POW(2, 2);  
      -> 4  
mysql> SELECT POW(2, -2);  
      -> 0.25
```

- RADIANS(X)

Returns the argument x , converted from degrees to radians. (Note that π radians equals 180 degrees.)

```
mysql> SELECT RADIANS(90);  
      -> 1.5707963267949
```

- RAND(), RAND(N)

Returns a random floating-point value v between 0 and 1 inclusive (that is, in the range $0 \leq v \leq 1.0$). If an integer argument N is specified, it is used as the seed value, which produces a repeatable sequence.

```
mysql> SELECT RAND();
```

```

-> 0.9233482386203
mysql> SELECT RAND(20);
-> 0.15888261251047
mysql> SELECT RAND(20);
-> 0.15888261251047
mysql> SELECT RAND();
-> 0.63553050033332
mysql> SELECT RAND();
-> 0.70100469486881
mysql> SELECT RAND(20);
-> 0.15888261251047

```

To obtain a random integer R in the range $i \leq R \leq j$, use the expression $\text{FLOOR}(i + \text{RAND}() * (j - i))$. For example, to obtain a random integer in the range of 7 to 12 inclusive, you could use the following statement:

```
SELECT FLOOR(7 + (RAND() * 5));
```

You cannot use a column with $\text{RAND}()$ values in an ORDER BY clause, because ORDER BY would evaluate the column multiple times. However, you can retrieve rows in random order like this:

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

$\text{ORDER BY RAND}()$ combined with LIMIT is useful for selecting a random sample from a set of rows:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d -> ORDER BY RAND();
```

Note that $\text{RAND}()$ in a WHERE clause is re-evaluated every time the WHERE is executed.

$\text{RAND}()$ is not meant to be a perfect random generator, but instead is a fast way to generate *ad hoc* random numbers which is portable between platforms for the same MySQL version.

- $\text{ROUND}(X)$, $\text{ROUND}(X,D)$

Returns the argument x , rounded to the nearest integer. With two arguments, returns x rounded to D decimal places. D can be negative to cause D digits left of the decimal point of the value x to become zero.

```
mysql> SELECT ROUND(-1.23);
-> -1
```

```
mysql> SELECT ROUND(-1.58);
      -> -2
mysql> SELECT ROUND(1.58);
      -> 2
mysql> SELECT ROUND(1.298, 1);
      -> 1.3
mysql> SELECT ROUND(1.298, 0);
      -> 1
mysql> SELECT ROUND(23.298, -1);
      -> 20
```

The return type is the same type as that of the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places).

Before MySQL 5.0.3, the behavior of `ROUND()` when the argument is halfway between two integers depends on the C library implementation. Different implementations round to the nearest even number, always up, always down, or always toward zero. If you need one kind of rounding, you should use a well-defined function such as `TRUNCATE()` or `FLOOR()` instead.

As of MySQL 5.0.3, `ROUND()` uses the precision math library for exact-value arguments when the first argument is a decimal value:

- For exact-value numbers, `ROUND()` uses the “round half up” or “round toward nearest” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the "round to nearest even" rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
```

| 3 | | 2 |
+-----+-----+

For more information, see [Chapter 21, Precision Math](#).

- SIGN(X)

Returns the sign of the argument as -1, 0, or 1, depending on whether x is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);  
      -> -1  
mysql> SELECT SIGN(0);  
      -> 0  
mysql> SELECT SIGN(234);  
      -> 1
```

- SIN(X)

Returns the sine of x , where x is given in radians.

```
mysql> SELECT SIN(PI());  
      -> 1.2246063538224e-16  
mysql> SELECT ROUND(SIN(PI()));  
      -> 0
```

- SQRT(X)

Returns the square root of a non-negative number x .

```
mysql> SELECT SQRT(4);  
      -> 2  
mysql> SELECT SQRT(20);  
      -> 4.4721359549996  
mysql> SELECT SQRT(-16);  
      -> NULL
```

- TAN(X)

Returns the tangent of x , where x is given in radians.

```
mysql> SELECT TAN(PI());  
      -> -1.2246063538224e-16  
mysql> SELECT TAN(PI()+1);  
      -> 1.5574077246549
```

- `TRUNCATE(X,D)`

Returns the number x , truncated to D decimal places. If D is 0, the result has no decimal point or fractional part. D can be negative to cause D digits left of the decimal point of the value x to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
      -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
      -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
      -> 1
mysql> SELECT TRUNCATE(-1.999,1);
      -> -1.9
mysql> SELECT TRUNCATE(122,-2);
      -> 100
mysql> SELECT TRUNCATE(10.28*100,0);
      -> 1028
```

All numbers are rounded toward zero.

12.5. Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See [Section 11.3, “Date and Time Types”](#), for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Here is an example that uses date functions. The following query selects all rows with a *date_col* value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
      -> WHERE DATE_SUB(CURDATE(), INTERVAL 30 DAY) <= date_col;
```

Note that the query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as `NOW()` within a single query always produce the same result (for our purposes a single query also includes a call to a stored routine or trigger and all sub-routines called by that routine/trigger). This principle also applies to `CURDATE()`, `CURTIME()`, `UTC_DATE()`, `UTC_TIME()`, `UTC_TIMESTAMP()`, and to any of their synonyms.

The `CURRENT_TIMESTAMP()`, `CURRENT_TIME()`, `CURRENT_DATE()`, and `FROM_UNIXTIME()` functions return values in the connection's current time zone, which is available as the value of the `time_zone` system variable. In addition, `UNIX_TIMESTAMP()` assumes that its argument is a datetime value in the current time zone. See [Section 5.11.8, “MySQL Server Time Zone Support”](#).

Some date functions can be used with “zero” dates or incomplete dates such as `'2001-11-00'`, whereas others cannot. Functions that extract parts of dates typically work with incomplete dates. For example:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
      -> 0, 0
```

Other functions expect complete dates and return NULL for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
      -> NULL
mysql> SELECT DAYNAME('2006-05-00');
      -> NULL
```

- `ADDDATE(date,INTERVAL expr unit), ADDDATE(expr,days)`

When invoked with the `INTERVAL` form of the second argument, `ADDDATE()` is a synonym for `DATE_ADD()`. The related function `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL unit` argument, see the discussion for `DATE_ADD()`.

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
      -> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
      -> '1998-02-02'
```

When invoked with the `days` form of the second argument, MySQL treats it as an integer number of days to be added to `expr`.

```
mysql> SELECT ADDDATE('1998-01-02', 31);
      -> '1998-02-02'
```

- `ADDTIME(expr1,expr2)`

`ADDTIME()` adds `expr2` to `expr1` and returns the result. `expr1` is a time or datetime expression, and `expr2` is a time expression.

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999',
      ->                '1 1:1:1.000002');
      -> '1998-01-02 01:01:01.000001'
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
      -> '03:00:01.999997'
```

- `CONVERT_TZ(dt,from_tz,to_tz)`

`CONVERT_TZ()` converts a datetime value `dt` from the time zone given by `from_tz` to the time zone given by `to_tz` and returns the resulting value. Time zones are specified as described in [Section 5.11.8, “MySQL Server](#)

[Time Zone Support](#)". This function returns NULL if the arguments are invalid.

If the value falls out of the supported range of the TIMESTAMP type when converted from *from_tz* to UTC, no conversion occurs. The TIMESTAMP range is described in [Section 11.1.2, "Overview of Date and Time Types"](#).

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00', 'GMT', 'MET');
      -> '2004-01-01 13:00:00'
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00', '+00:00', '+10:00')
      -> '2004-01-01 22:00:00'
```

Note: To use named time zones such as 'MET' or 'Europe/Moscow', the time zone tables must be properly set up. See [Section 5.11.8, "MySQL Server Time Zone Support"](#), for instructions.

If you intend to use CONVERT_TZ() while other tables are locked with LOCK TABLES, you must also lock the mysql.time_zone_name table.

- CURDATE()

Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
      -> '1997-12-15'
mysql> SELECT CURDATE() + 0;
      -> 19971215
```

- CURRENT_DATE, CURRENT_DATE()

CURRENT_DATE and CURRENT_DATE() are synonyms for CURDATE().

- CURTIME()

Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURTIME();
      -> '23:50:26'
mysql> SELECT CURTIME() + 0;
      -> 235026
```

- CURRENT_TIME, CURRENT_TIME()

CURRENT_TIME and CURRENT_TIME() are synonyms for CURTIME().

- CURRENT_TIMESTAMP, CURRENT_TIMESTAMP()

CURRENT_TIMESTAMP and CURRENT_TIMESTAMP() are synonyms for NOW().

- DATE(expr)

Extracts the date part of the date or datetime expression *expr*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
      -> '2003-12-31'
```

- DATEDIFF(expr1,expr2)

DATEDIFF() returns *expr1* – *expr2* expressed as a value in days from one date to the other. *expr1* and *expr2* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
      -> 1
mysql> SELECT DATEDIFF('1997-11-30 23:59:59','1997-12-31');
      -> -31
```

- DATE_ADD(date,INTERVAL *expr unit*), DATE_SUB(date,INTERVAL *expr unit*)

These functions perform date arithmetic. *date* is a DATETIME or DATE value specifying the starting date. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a '-' for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted.

The INTERVAL keyword and the *unit* specifier are not case sensitive.

The following table shows the expected form of the *expr* argument for each *unit* value.

<i>unit</i> Value	Expected <i>expr</i> Format
MICROSECOND	MICROSECONDS

SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

The values QUARTER and WEEK are available beginning with MySQL 5.0.0.

MySQL allows any punctuation delimiter in the *expr* format. Those shown in the table are the suggested delimiters. If the *date* argument is a DATE value and your calculations involve only YEAR, MONTH, and DAY parts (that is, no time parts), the result is a DATE value. Otherwise, the result is a DATETIME value.

Date arithmetic also can be performed using INTERVAL together with the + or - operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

INTERVAL *expr* *unit* is allowed on either side of the + operator if the expression on the other side is a date or datetime value. For the - operator, INTERVAL *expr* *unit* is allowed only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```

mysql> SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
      -> '1998-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
      -> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
      -> '1997-12-31 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
      -> INTERVAL 1 SECOND);
      -> '1998-01-01 00:00:00'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
      -> INTERVAL 1 DAY);
      -> '1998-01-01 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
      -> INTERVAL '1:1' MINUTE_SECOND);
      -> '1998-01-01 00:01:00'
mysql> SELECT DATE_SUB('1998-01-01 00:00:00',
      -> INTERVAL '1 1:1:1' DAY_SECOND);
      -> '1997-12-30 22:58:59'
mysql> SELECT DATE_ADD('1998-01-01 00:00:00',
      -> INTERVAL '-1 10' DAY_HOUR);
      -> '1997-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
      -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
      -> INTERVAL '1.999999' SECOND_MICROSECOND);
      -> '1993-01-01 00:00:01.000001'

```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the *unit* keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a *unit* of `DAY_SECOND`, the value of *expr* is expected to have days, hours, minutes, and seconds parts. If you specify a value like `'1:10'`, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `'1:10' DAY_SECOND` is interpreted in such a way that it is equivalent to `'1:10' MINUTE_SECOND`. This is analogous to the way that MySQL interprets `TIME` values as representing elapsed time rather than as a time of day.

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```

mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
      -> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
      -> '1999-01-01 01:00:00'

```

If you add MONTH, YEAR_MONTH, or YEAR and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);  
      -> '1998-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as '2006-07-00' or badly malformed dates:

```
mysql> SELECT DATE_ADD('2006-07-00', INTERVAL 1 DAY);  
      -> NULL  
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;  
      -> NULL
```

- DATE_FORMAT(*date*,*format*)

Formats the *date* value according to the *format* string.

The following specifiers may be used in the *format* string. The '%' character is required before format specifier characters.

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)

%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X
%v	Week (01..53), where Monday is the first day of the week; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %v
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal '%' character
%x	x, for any 'x' not listed above

Ranges for the month and day specifiers begin with zero due to the fact that MySQL allows the storing of incomplete dates such as '2004-00-00'.

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
-> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%D %y %a %d %m %b %j');
-> '4th 97 Sat 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%H %k %I %r %T %S %w');
```

```
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
-> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
-> '00'
```

- DAY(date)

DAY() is a synonym for DAYOFMONTH().

- DAYNAME(date)

Returns the name of the weekday for *date*.

```
mysql> SELECT DAYNAME('1998-02-05');
-> 'Thursday'
```

- DAYOFMONTH(date)

Returns the day of the month for *date*, in the range 0 to 31.

```
mysql> SELECT DAYOFMONTH('1998-02-03');
-> 3
```

- DAYOFWEEK(date)

Returns the weekday index for *date* (1 = Sunday, 2 = Monday, ..., 7 = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
-> 3
```

- DAYOFYEAR(date)

Returns the day of the year for *date*, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('1998-02-03');
-> 34
```

- EXTRACT(unit FROM *date*)

The EXTRACT() function uses the same kinds of unit specifiers as DATE_ADD() or DATE_SUB(), but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
-> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
-> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '1999-07-02 01:02:03');
-> 20102
mysql> SELECT EXTRACT(MICROSECOND
-> FROM '2003-01-02 10:30:00.00123');
-> 123
```

- FROM_DAYS(N)

Given a day number *N*, returns a DATE value.

```
mysql> SELECT FROM_DAYS(729669);
-> '1997-10-07'
```

Use FROM_DAYS() with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See [Section 12.6, “What Calendar Is Used By MySQL?”](#).

- FROM_UNIXTIME(unix_timestamp),
FROM_UNIXTIME(unix_timestamp,format)

Returns a representation of the *unix_timestamp* argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. *unix_timestamp* is an internal timestamp value such as is produced by the UNIX_TIMESTAMP() function.

If *format* is given, the result is formatted according to the *format* string, which is used the same way as listed in the entry for the DATE_FORMAT() function.

```
mysql> SELECT FROM_UNIXTIME(875996580);
-> '1997-10-04 22:23:00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
-> 19971004222300
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
-> '%Y %D %M %h:%i:%s %x');
-> '2003 6th August 06:22:58 2003'
```

Note: If you use UNIX_TIMESTAMP() and FROM_UNIXTIME() to convert between TIMESTAMP values and Unix timestamp values, the conversion is

lossy because the mapping is not one-to-one in both directions. For details, see the description of the `UNIX_TIMESTAMP()` function.

- `GET_FORMAT(DATE | TIME | DATETIME, 'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL')`

Returns a format string. This function is useful in combination with the `DATE_FORMAT()` and the `STR_TO_DATE()` functions.

The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the `DATE_FORMAT()` function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
<code>GET_FORMAT(DATE, 'USA')</code>	<code>'%m.%d.%Y'</code>
<code>GET_FORMAT(DATE, 'JIS')</code>	<code>'%Y-%m-%d'</code>
<code>GET_FORMAT(DATE, 'ISO')</code>	<code>'%Y-%m-%d'</code>
<code>GET_FORMAT(DATE, 'EUR')</code>	<code>'%d.%m.%Y'</code>
<code>GET_FORMAT(DATE, 'INTERNAL')</code>	<code>'%Y%m%d'</code>
<code>GET_FORMAT(DATETIME, 'USA')</code>	<code>'%Y-%m-%d-%H.%i.%s'</code>
<code>GET_FORMAT(DATETIME, 'JIS')</code>	<code>'%Y-%m-%d %H:%i:%s'</code>
<code>GET_FORMAT(DATETIME, 'ISO')</code>	<code>'%Y-%m-%d %H:%i:%s'</code>
<code>GET_FORMAT(DATETIME, 'EUR')</code>	<code>'%Y-%m-%d-%H.%i.%s'</code>
<code>GET_FORMAT(DATETIME, 'INTERNAL')</code>	<code>'%Y%m%d%H%i%s'</code>
<code>GET_FORMAT(TIME, 'USA')</code>	<code>'%h:%i:%s %p'</code>
<code>GET_FORMAT(TIME, 'JIS')</code>	<code>'%H:%i:%s'</code>
<code>GET_FORMAT(TIME, 'ISO')</code>	<code>'%H:%i:%s'</code>
<code>GET_FORMAT(TIME, 'EUR')</code>	<code>'%H.%i.%S'</code>
<code>GET_FORMAT(TIME, 'INTERNAL')</code>	<code>'%H%i%s'</code>

`TIMESTAMP` can also be used as the first argument to `GET_FORMAT()`, in which case the function returns the same values as for `DATETIME`.

```
mysql> SELECT DATE_FORMAT('2003-10-03', GET_FORMAT( DATE, 'EUR' ));
-> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003', GET_FORMAT( DATE, 'USA' ));
-> '2003-10-31'
```

- `HOUR(time)`

Returns the hour for *time*. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');
      -> 10
mysql> SELECT HOUR('272:59:59');
      -> 272
```

- LAST_DAY(date)

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
      -> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
      -> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
      -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
      -> NULL
```

- LOCALTIME, LOCALTIME()

LOCALTIME and LOCALTIME() are synonyms for NOW().

- LOCALTIMESTAMP, LOCALTIMESTAMP()

LOCALTIMESTAMP and LOCALTIMESTAMP() are synonyms for NOW().

- MAKEDATE(year,dayofyear)

Returns a date, given year and day-of-year values. *dayofyear* must be greater than 0 or the result is NULL.

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
      -> '2001-01-31', '2001-02-01'
mysql> SELECT MAKEDATE(2001,365), MAKEDATE(2004,365);
      -> '2001-12-31', '2004-12-30'
mysql> SELECT MAKEDATE(2001,0);
      -> NULL
```

- MAKETIME(hour,minute,second)

Returns a time value calculated from the *hour*, *minute*, and *second* arguments.

```
mysql> SELECT MAKETIME(12,15,30);  
      -> '12:15:30'
```

- MICROSECOND(expr)

Returns the microseconds from the time or datetime expression *expr* as a number in the range from 0 to 999999.

```
mysql> SELECT MICROSECOND('12:00:00.123456');  
      -> 123456  
mysql> SELECT MICROSECOND('1997-12-31 23:59:59.000010');  
      -> 10
```

- MINUTE(time)

Returns the minute for *time*, in the range 0 to 59.

```
mysql> SELECT MINUTE('98-02-03 10:05:03');  
      -> 5
```

- MONTH(date)

Returns the month for *date*, in the range 0 to 12.

```
mysql> SELECT MONTH('1998-02-03');  
      -> 2
```

- MONTHNAME(date)

Returns the full name of the month for *date*.

```
mysql> SELECT MONTHNAME('1998-02-05');  
      -> 'February'
```

- NOW()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT NOW();
```

```

-> '1997-12-15 23:50:26'
mysql> SELECT NOW() + 0;
-> 19971215235026

```

`NOW()` returns a constant time that indicates the time at which the statement began to execute. (Within a stored routine or trigger, `NOW()` returns the time at which the routine or triggering statement began to execute.) This differs from the behavior for `SYSDATE()`, which returns the exact time at which it executes as of MySQL 5.0.13.

```

mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 |          0 | 2006-04-12 13:47:36 |
+-----+-----+-----+

```

```

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 |          0 | 2006-04-12 13:47:46 |
+-----+-----+-----+

```

See the description for `SYSDATE()` for additional information about the differences between the two functions.

- `PERIOD_ADD(P,N)`

Adds N months to period P (in the format `YYMM` or `YYYYMM`). Returns a value in the format `YYYYMM`. Note that the period argument P is *not* a date value.

```

mysql> SELECT PERIOD_ADD(9801,2);
-> 199803

```

- `PERIOD_DIFF(P1,P2)`

Returns the number of months between periods $P1$ and $P2$. $P1$ and $P2$ should be in the format `YYMM` or `YYYYMM`. Note that the period arguments $P1$ and $P2$ are *not* date values.

```

mysql> SELECT PERIOD_DIFF(9802,199703);
-> 11

```

- `QUARTER(date)`

Returns the quarter of the year for *date*, in the range 1 to 4.

```
mysql> SELECT QUARTER('98-04-01');  
-> 2
```

- `SECOND(time)`

Returns the second for *time*, in the range 0 to 59.

```
mysql> SELECT SECOND('10:05:03');  
-> 3
```

- `SEC_TO_TIME(seconds)`

Returns the *seconds* argument, converted to hours, minutes, and seconds, as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT SEC_TO_TIME(2378);  
-> '00:39:38'  
mysql> SELECT SEC_TO_TIME(2378) + 0;  
-> 3938
```

- `STR_TO_DATE(str,format)`

This is the inverse of the `DATE_FORMAT()` function. It takes a string *str* and a format string *format*. `STR_TO_DATE()` returns a `DATETIME` value if the format string contains both date and time parts, or a `DATE` or `TIME` value if the string contains only date or time parts.

The date, time, or datetime values contained in *str* should be given in the format indicated by *format*. For the specifiers that can be used in *format*, see the `DATE_FORMAT()` function description. If *str* contains an illegal date, time, or datetime value, `STR_TO_DATE()` returns `NULL`. Starting from MySQL 5.0.3, an illegal value also produces a warning.

Range checking on the parts of date values is as described in [Section 11.3.1, “The DATETIME, DATE, and TIMESTAMP Types”](#). This means, for example, that “zero” dates or dates with part values of 0 are allowed unless the SQL mode is set to disallow such values.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
        -> '2004-04-31'
```

- SUBDATE(date,INTERVAL *expr unit*), SUBDATE(*expr,days*)

When invoked with the INTERVAL form of the second argument, SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL *unit* argument, see the discussion for DATE_ADD().

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
```

The second form allows the use of an integer value for *days*. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression *expr*.

```
mysql> SELECT SUBDATE('1998-01-02 12:00:00', 31);
        -> '1997-12-02 12:00:00'
```

Note: You cannot use format "%X%V" to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, then you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
        -> '2004-10-18'
```

- SUBTIME(*expr1,expr2*)

SUBTIME() returns *expr1* – *expr2* expressed as a value in the same format as *expr1*. *expr1* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999', '1 1:1:1.000000');
        -> '1997-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
        -> '-00:59:59.999999'
```

- SYSDATE()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

As of MySQL 5.0.13, `SYSDATE()` returns the time at which it executes. This differs from the behavior for `NOW()`, which returns a constant time that indicates the time at which the statement began to execute. (Within a stored routine or trigger, `NOW()` returns the time at which the routine or triggering statement began to execute.)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 |          0 | 2006-04-12 13:47:36 |
+-----+-----+-----+
```

```
mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 |          0 | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`.

Because `SYSDATE()` can return different values even within the same statement, and is not affected by `SET TIMESTAMP`, it is non-deterministic and therefore unsafe for replication. If that is a problem, you can start the server with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`.

- `TIME(expr)`

Extracts the time part of the time or datetime expression *expr* and returns it as a string.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
-> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
-> '01:02:03.000123'
```

- `TIMEDIFF(expr1,expr2)`

`TIMEDIFF()` returns $expr1 - expr2$ expressed as a time value. $expr1$ and $expr2$ are time or date-and-time expressions, but both must be of the same type.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
->                    '2000:01:01 00:00:00.000001');
-> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
->                    '1997-12-30 01:01:01.000002');
-> '46:58:57.999999'
```

- `TIMESTAMP(expr)`, `TIMESTAMP(expr1,expr2)`

With a single argument, this function returns the date or datetime expression $expr$ as a datetime value. With two arguments, it adds the time expression $expr2$ to the date or datetime expression $expr1$ and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
-> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
-> '2004-01-01 00:00:00'
```

- `TIMESTAMPADD(unit,interval,datetime_expr)`

Adds the integer expression $interval$ to the date or datetime expression $datetime_expr$. The unit for $interval$ is given by the $unit$ argument, which should be one of the following values: `FRAC_SECOND`, `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, or `YEAR`.

The $unit$ value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` and `SQL_TSI_DAY` both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
-> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
-> '2003-01-09'
```

`TIMESTAMPADD()` is available as of MySQL 5.0.0.

- `TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)`

Returns the integer difference between the date or datetime expressions *datetime_expr1* and *datetime_expr2*. The unit for the result is given by the *unit* argument. The legal values for *unit* are the same as those listed in the description of the `TIMESTAMPADD()` function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH, '2003-02-01', '2003-05-01');
-> 3
mysql> SELECT TIMESTAMPDIFF(YEAR, '2002-05-01', '2001-01-01');
-> -1
```

`TIMESTAMPDIFF()` is available as of MySQL 5.0.0.

- `TIME_FORMAT(time,format)`

This is used like the `DATE_FORMAT()` function, but the *format* string may contain format specifiers only for hours, minutes, and seconds. Other specifiers produce a `NULL` value or 0.

If the *time* value contains an hour part that is greater than 23, the `%H` and `%k` hour format specifiers produce a value larger than the usual range of 0..23. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
-> '100 100 04 04 4'
```

- `TIME_TO_SEC(time)`

Returns the *time* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
-> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
-> 2378
```

- `TO_DAYS(date)`

Given a date *date*, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
-> 728779
mysql> SELECT TO_DAYS('1997-10-07');
-> 729669
```

`TO_DAYS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See [Section 12.6, “What Calendar Is Used By MySQL?”](#), for details.

Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in [Section 11.3, “Date and Time Types”](#). For example, `'1997-10-07'` and `'97-10-07'` are seen as identical dates:

```
mysql> SELECT TO_DAYS('1997-10-07'), TO_DAYS('97-10-07');
-> 729669, 729669
```

- `UNIX_TIMESTAMP()`, `UNIX_TIMESTAMP(date)`

If called with no argument, returns a Unix timestamp (seconds since `'1970-01-01 00:00:00'` UTC) as an unsigned integer. If `UNIX_TIMESTAMP()` is called with a *date* argument, it returns the value of the argument as seconds since `'1970-01-01 00:00:00'` UTC. *date* may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format `YYMMDD` or `YYYYMMDD`. The server interprets *date* as a value in the current time zone and converts it to an internal value in UTC. Clients can set their time zone as described in [Section 5.11.8, “MySQL Server Time Zone Support”](#).

```
mysql> SELECT UNIX_TIMESTAMP();
-> 882226357
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
-> 875996580
```

When `UNIX_TIMESTAMP` is used on a `TIMESTAMP` column, the function returns the internal timestamp value directly, with no implicit “string-to-Unix-timestamp” conversion. If you pass an out-of-range date to `UNIX_TIMESTAMP()`, it returns `0`.

Note: If you use `UNIX_TIMESTAMP()` and `FROM_UNIXTIME()` to convert between `TIMESTAMP` values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For example, due to conventions for local time zone changes, it is possible for two `UNIX_TIMESTAMP()` to map two `TIMESTAMP` values to the same Unix timestamp value. `FROM_UNIXTIME()` will map that value back to only one of

the original `TIMESTAMP` values. Here is an example, using `TIMESTAMP` values in the CET time zone:

```
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+-----+
|                               1111885200 |
+-----+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+-----+
|                               1111885200 |
+-----+
mysql> SELECT FROM_UNIXTIME(1111885200);
+-----+
| FROM_UNIXTIME(1111885200) |
+-----+
| 2005-03-27 03:00:00      |
+-----+
```

If you want to subtract `UNIX_TIMESTAMP()` columns, you might want to cast the result to signed integers. See [Section 12.8, “Cast Functions and Operators”](#).

- `UTC_DATE`, `UTC_DATE()`

Returns the current UTC date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
-> '2003-08-14', 20030814
```

- `UTC_TIME`, `UTC_TIME()`

Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
-> '18:07:53', 180753
```

- `UTC_TIMESTAMP`, `UTC_TIMESTAMP()`

Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
        -> '2003-08-14 18:08:04', 20030814180804
```

- WEEK(date[,mode])

This function returns the week number for *date*. The two-argument form of WEEK() allows you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the *mode* argument is omitted, the value of the default_week_format system variable is used. See [Section 5.2.2, “Server System Variables”](#).

The following table describes how the *mode* argument works.

	First day		
Mode	of week	Range	Week 1 is the first week ...
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with more than 3 days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with more than 3 days this year
4	Sunday	0-53	with more than 3 days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with more than 3 days this year
7	Monday	1-53	with a Monday in this year

```
mysql> SELECT WEEK('1998-02-20');
        -> 7
mysql> SELECT WEEK('1998-02-20', 0);
        -> 7
mysql> SELECT WEEK('1998-02-20', 1);
        -> 8
mysql> SELECT WEEK('1998-12-31', 1);
        -> 53
```

Note that if a date falls in the last week of the previous year, MySQL

returns 0 if you do not use 2, 3, 6, or 7 as the optional *mode* argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01', 0);  
      -> 2000, 0
```

One might argue that MySQL should return 52 for the `WEEK()` function, because the given date actually occurs in the 52nd week of 1999. We decided to return 0 instead because we want the function to return “the week number in the given year.” This makes use of the `WEEK()` function reliable when combined with other functions that extract a date part from a date.

If you would prefer the result to be evaluated with respect to the year that contains the first day of the week for the given date, use 0, 2, 5, or 7 as the optional *mode* argument.

```
mysql> SELECT WEEK('2000-01-01', 2);  
      -> 52
```

Alternatively, use the `YEARWEEK()` function:

```
mysql> SELECT YEARWEEK('2000-01-01');  
      -> 199952  
mysql> SELECT MID(YEARWEEK('2000-01-01'), 5, 2);  
      -> '52'
```

- `WEEKDAY(date)`

Returns the weekday index for *date* (0 = Monday, 1 = Tuesday, ... 6 = Sunday).

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');  
      -> 1  
mysql> SELECT WEEKDAY('1997-11-05');  
      -> 2
```

- `WEEKOFYEAR(date)`

Returns the calendar week of the date as a number in the range from 1 to 53. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(date,3)`.

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
```

-> 8

- YEAR(date)

Returns the year for *date*, in the range 1000 to 9999, or 0 for the “zero” date.

```
mysql> SELECT YEAR('98-02-03');  
-> 1998
```

- YEARWEEK(date), YEARWEEK(date,*start*)

Returns year and week for a date. The *start* argument works exactly like the *start* argument to WEEK(). The year in the result may be different from the year in the date argument for the first and the last week of the year.

```
mysql> SELECT YEARWEEK('1987-01-01');  
-> 198653
```

Note that the week number is different from what the WEEK() function would return (0) for optional arguments 0 or 1, as WEEK() then returns the week in the context of the given year.

12.6. What Calendar Is Used By MySQL?

MySQL uses what is known as a *proleptic Gregorian calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred:

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are non-existent.

A calendar applied to dates when it wasn't actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL `DATE` or `DATETIME` values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

12.7. Full-Text Search Functions

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])
```

```
search_modifier: { IN BOOLEAN MODE | WITH QUERY EXPANSION }
```

MySQL has support for full-text indexing and searching:

- A full-text index in MySQL is an index of type `FULLTEXT`.
- Full-text indexes can be used only with `MyISAM` tables, and can be created only for `CHAR`, `VARCHAR`, or `TEXT` columns.
- A `FULLTEXT` index definition can be given in the `CREATE TABLE` statement when a table is created, or added later using `ALTER TABLE` or `CREATE INDEX`.
- For large datasets, it is much faster to load your data into a table that has no `FULLTEXT` index and then create the index after that, than to load data into a table that has an existing `FULLTEXT` index.

Full-text searching is performed using `MATCH() ... AGAINST` syntax. `MATCH()` takes a comma-separated list that names the columns to be searched. `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a literal string, not a variable or a column name. There are three types of full-text searches:

- A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Common words such as “some” or “then” are stopwords and do not match if present in the search string. The `IN BOOLEAN MODE` modifier specifies a boolean search. For more information, see [Section 12.7.1, “Boolean Full-Text Searches”](#).
- A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators. The stopword list applies. In addition, words that are present in more than 50% of the rows are considered common and do not match. Full-text searches

are natural language searches if no modifier is given.

- A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The `WITH QUERY EXPANSION` modifier specifies a query expansion search. For more information, see [Section 12.7.2, “Full-Text Searches with Query Expansion”](#).

Constraints on full-text searching are listed in [Section 12.7.4, “Full-Text Restrictions”](#).

```
mysql> CREATE TABLE articles (  
-> id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,  
-> title VARCHAR(200),  
-> body TEXT,  
-> FULLTEXT (title,body)  
-> );  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO articles (title,body) VALUES  
-> ('MySQL Tutorial','DBMS stands for DataBase ...'),  
-> ('How To Use MySQL Well','After you went through a ...'),  
-> ('Optimizing MySQL','In this tutorial we will show ...'),  
-> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),  
-> ('MySQL vs. YourSQL','In the following database comparison ..  
-> ('MySQL Security','When configured properly, MySQL ...');  
Query OK, 6 rows affected (0.00 sec)  
Records: 6 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM articles  
-> WHERE MATCH (title,body) AGAINST ('database');  
+-----+-----+-----+-----+  
| id | title | body |  
+-----+-----+-----+-----+  
| 5 | MySQL vs. YourSQL | In the following database comparison ...  
| 1 | MySQL Tutorial | DBMS stands for DataBase ...  
+-----+-----+-----+-----+  
2 rows in set (0.00 sec)
```

The `MATCH()` function performs a natural language search for a string against a *text collection*. A collection is a set of one or more columns included in a `FULLTEXT` index. The search string is given as the argument to `AGAINST()`. For each row in the table, `MATCH()` returns a relevance value; that is, a similarity

measure between the search string and the text in that row in the columns named in the `MATCH()` list.

By default, the search is performed in case-insensitive fashion. However, you can perform a case-sensitive full-text search by using a binary collation for the indexed columns. For example, a column that uses the `latin1` character set of can be assigned a collation of `latin1_bin` to make it case sensitive for full-text searches.

When `MATCH()` is used in a `WHERE` clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first. Relevance values are non-negative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row, the number of unique words in that row, the total number of words in the collection, and the number of documents (rows) that contain a particular word.

For natural-language full-text searches, it is a requirement that the columns named in the `MATCH()` function be the same columns included in some `FULLTEXT` index in your table. For the preceding query, note that the columns named in the `MATCH()` function (`title` and `body`) are the same as those named in the definition of the `article` table's `FULLTEXT` index. If you wanted to search the `title` or `body` separately, you would need to create separate `FULLTEXT` indexes for each column.

It is also possible to perform a boolean search or a search with query expansion. These search types are described in [Section 12.7.1, “Boolean Full-Text Searches”](#), and [Section 12.7.2, “Full-Text Searches with Query Expansion”](#).

The preceding example is a basic illustration that shows how to use the `MATCH()` function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the `SELECT` statement includes neither `WHERE` nor `ORDER BY` clauses:

```
mysql> SELECT id, MATCH (title,body) AGAINST ('Tutorial')
      -> FROM articles;
```

```
+-----+-----+
| id | MATCH (title,body) AGAINST ('Tutorial') |
+-----+-----+
| 1 | 0.65545833110809 |
| 2 | 0 |
| 3 | 0.66266459226608 |
```

```

| 4 | | 0 |
| 5 | | 0 |
| 6 | | 0 |
+---+-----+
6 rows in set (0.00 sec)

```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, you should specify `MATCH()` twice: once in the `SELECT` list and once in the `WHERE` clause. This causes no additional overhead, because the MySQL optimizer notices that the two `MATCH()` calls are identical and invokes the full-text search code only once.

```

mysql> SELECT id, body, MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root') AS score
-> FROM articles WHERE MATCH (title,body) AGAINST
-> ('Security implications of running MySQL as root');
+---+-----+-----+
| id | body | score |
+---+-----+-----+
| 4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
| 6 | When configured properly, MySQL ... | 1.3114095926285 |
+---+-----+-----+
2 rows in set (0.00 sec)

```

The MySQL `FULLTEXT` implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes (`'`), but not more than one in a row. This means that `aaa'bbb` is regarded as one word, but `aaa''bbb` is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the `FULLTEXT` parser; `'aaa'bbb'` would be parsed as `aaa'bbb`.

The `FULLTEXT` parser determines where words start and end by looking for certain delimiter characters; for example, `'` (space), `,` (comma), and `.` (period). If words are not separated by delimiters (as in, for example, Chinese), the `FULLTEXT` parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a `FULLTEXT` index, you must preprocess them so that they are separated by some arbitrary delimiter such as `'`.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words

that are found by full-text searches is four characters.

- Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overwritten by a user-defined list.

The default stopword list is given in [Section 12.7.3, “Full-Text Stopwords”](#). The default minimum word length and stopword list can be changed as described in [Section 12.7.5, “Fine-Tuning MySQL Full-Text Search”](#).

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Consequently, a word that is present in many documents has a lower weight (and may even have a zero weight), because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row.

Such a technique works best with large collections (in fact, it was carefully tuned this way). For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results. For example, although the word “MySQL” is present in every row of the `articles` table shown earlier, a search for the word produces no results:

```
mysql> SELECT * FROM articles
      -> WHERE MATCH (title,body) AGAINST ('MySQL');
Empty set (0.00 sec)
```

The search result is empty because the word “MySQL” is present in at least 50% of the rows. As such, it is effectively treated as a stopword. For large datasets, this is the most desirable behavior: A natural language query should not return every second row from a 1GB table. For small datasets, it may be less desirable.

A word that matches half of the rows in a table is less likely to locate relevant documents. In fact, it most likely finds plenty of irrelevant documents. We all know this happens far too often when we are trying to find something on the Internet with a search engine. It is with this reasoning that rows containing the word are assigned a low semantic value for *the particular dataset in which they occur*. A given word may exceed the 50% threshold in one dataset but not another.

The 50% threshold has a significant implication when you first try full-text searching to see how it works: If you create a table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results. Be sure to insert at least three rows, and preferably many more. Users who need to bypass the 50% limitation can use the boolean search mode; see [Section 12.7.1, “Boolean Full-Text Searches”](#).

12.7.1. Boolean Full-Text Searches

MySQL can perform boolean full-text searches using the `IN BOOLEAN MODE` modifier:

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
      -> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+-----+-----+-----+-----+
| id | title                | body                |
+-----+-----+-----+-----+
| 1  | MySQL Tutorial       | DBMS stands for DataBase ...
| 2  | How To Use MySQL Well | After you went through a ...
| 3  | Optimizing MySQL    | In this tutorial we will show ...
| 4  | 1001 MySQL Tricks   | 1. Never run mysqld as root. 2. ...
| 6  | MySQL Security       | When configured properly, MySQL ...
+-----+-----+-----+-----+
```

The `+` and `-` operators indicate that a word is required to be present or absent, respectively, for a match to occur. Thus, this query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL”.

Boolean full-text searches have these characteristics:

- They do not use the 50% threshold.
- They do not automatically sort rows in order of decreasing relevance. You can see this from the preceding query result: The row with the highest relevance is the one that contains “MySQL” twice, but it is listed last, not first.
- They can work even without a `FULLTEXT` index, although a search executed in this fashion would be quite slow.
- The minimum and maximum word length full-text parameters apply.

- The stopword list applies.

The boolean full-text search capability supports the following operators:

- +

A leading plus sign indicates that this word *must* be present in each row that is returned.

- -

A leading minus sign indicates that this word must *not* be present in any of the rows that are returned.

Note: The - operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by - returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

- (no operator)

By default (when neither + nor - is specified) the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() ... AGAINST()` without the `IN BOOLEAN MODE` modifier.

- > <

These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The > operator increases the contribution and the < operator decreases it. See the example following this list.

- ()

Parentheses group words into subexpressions. Parenthesized groups can be nested.

- ~

A leading tilde acts as a negation operator, causing the word's contribution

to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the - operator.

- *

The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it should be *appended* to the word to be affected. Words match if they begin with the word preceding the * operator.

- "

A phrase that is enclosed within double quote (“”) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words, performs a search in the FULLTEXT index for the words. Prior to MySQL 5.0.3, the engine then performed a substring search for the phrase in the records that were found, so the match must include non-word characters in the phrase. As of MySQL 5.0.3, non-word characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, "test phrase" matches "test, phrase" in MySQL 5.0.3, but not before.

If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The following examples demonstrate some search strings that use boolean full-text operators:

- 'apple banana'

Find rows that contain at least one of the two words.

- '+apple +juice'

Find rows that contain both words.

- '+apple macintosh'

Find rows that contain the word “apple”, but rank rows higher if they also contain “macintosh”.

- '+apple -macintosh'

Find rows that contain the word “apple” but not “macintosh”.

- '+apple ~macintosh'

Find rows that contain the word “apple”, but if the row also contains the word “macintosh”, rate it lower than if row does not. This is “softer” than a search for '+apple -macintosh', for which the presence of “macintosh” causes the row not to be returned at all.

- '+apple +(>turnover <strudel)'

Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.

- 'apple*'

Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

- '"some words"'

Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”). Note that the ‘”’ characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotes that enclose the search string itself.

12.7.2. Full-Text Searches with Query Expansion

Full-text search supports query expansion (and in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL”, “Oracle”, “DB2”, and “RDBMS” all are phrases that

should match “databases” and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL”, the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database”. The following example shows this difference:

```
mysql> SELECT * FROM articles
      -> WHERE MATCH (title,body) AGAINST ('database');
+----+-----+-----+
| id | title                | body
+----+-----+-----+
|  5 | MySQL vs. YourSQL   | In the following database comparison ...
|  1 | MySQL Tutorial      | DBMS stands for DataBase ...
+----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM articles
      -> WHERE MATCH (title,body)
      -> AGAINST ('database' WITH QUERY EXPANSION);
+----+-----+-----+
| id | title                | body
+----+-----+-----+
|  1 | MySQL Tutorial      | DBMS stands for DataBase ...
|  5 | MySQL vs. YourSQL   | In the following database comparison ...
|  3 | Optimizing MySQL    | In this tutorial we will show ...
+----+-----+-----+
3 rows in set (0.00 sec)
```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret”. A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.

Note: Because blind query expansion tends to increase noise significantly by returning non-relevant documents, it is meaningful to use only when a search phrase is rather short.

12.7.3. Full-Text Stopwords

The following table shows the default list of full-text stopwords.

a's	able	about	above	according
accordingly	across	actually	after	afterwards
again	against	ain't	all	allow
allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway
anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co
com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during
each	edu	eg	eight	either
else	elsewhere	enough	entirely	especially
et	etc	even	ever	every

everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few
fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten
greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored
immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	knows	known	last	lately
later	latter	latterly	least	less
lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new

next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems
seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends
th	than	thank	thanks	thanx
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think

third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly
try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who
who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
would	wouldn't	yes	yet	you
you'd	you'll	you're	you've	your
yours	yourself	yourselves	zero	

12.7.4. Full-Text Restrictions

- Full-text searches are supported for MyISAM tables only.
- Full-text searches can be used with most multi-byte character sets. The exception is that for Unicode, the utf8 character set can be used, but not the ucs2 character set.
- Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the FULLTEXT parser *cannot determine where words*

begin and end in these and other such languages. The implications of this and some workarounds for the problem are described in [Section 12.7, “Full-Text Search Functions”](#).

- Although the use of multiple character sets within a single table is supported, all columns in a FULLTEXT index must use the same character set and collation.
- The MATCH() column list must match exactly the column list in some FULLTEXT index definition for the table, unless this MATCH() is IN BOOLEAN MODE. Boolean-mode searches can be done on non-indexed columns, although they are likely to be slow.
- The argument to AGAINST() must be a constant string.

12.7.5. Fine-Tuning MySQL Full-Text Search

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See [Section 2.9, “MySQL Installation Using a Source Distribution”](#).

Note that full-text search is carefully tuned for the most effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do not alter the MySQL sources unless you know what you are doing.*

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the FULLTEXT indexes in your tables. Instructions for doing this are given at the end of this section.

- The minimum and maximum lengths of words to be indexed are defined by the ft_min_word_len and ft_max_word_len system variables. (See [Section 5.2.2, “Server System Variables”](#).) The default minimum value is four characters; the default maximum is version dependent. If you change either value, you must rebuild your FULLTEXT indexes. For example, if you want three-character words to be searchable, you can set the

ft_min_word_len variable by putting the following lines in an option file:

```
[mysqld]
ft_min_word_len=3
```

Then you must restart the server and rebuild your FULLTEXT indexes. Note particularly the remarks regarding **myisamchk** in the instructions following this list.

- To override the default stopwords list, set the ft_stopword_file system variable. (See [Section 5.2.2, “Server System Variables”](#).) The variable value should be the pathname of the file containing the stopwords list, or the empty string to disable stopwords filtering. After changing the value of this variable or the contents of the stopwords file, restart the server and rebuild your FULLTEXT indexes.

The stopwords list is free-form. That is, you may use any non-alphanumeric character such as newline, space, or comma to separate stopwords. Exceptions are the underscore character (‘_’) and a single apostrophe (‘’’) which are treated as part of a word. The character set of the stopwords list is the server's default character set; see [Section 10.3.1, “Server Character Set and Collation”](#).

- The 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in myisam/ftdefs.h:

```
#define GWS_IN_USE GWS_PROB
```

Change that line to this:

```
#define GWS_IN_USE GWS_FREQ
```

Then recompile MySQL. There is no need to rebuild the indexes in this case. **Note:** By making this change, you *severely* decrease MySQL's ability to provide adequate relevance values for the MATCH() function. If you really need to search for such common words, it would be better to search using IN BOOLEAN MODE instead, which does not observe the 50% threshold.

- To change the operators used for boolean full-text searches, set the ft_boolean_syntax system variable. This variable can be changed while

the server is running, but you must have the SUPER privilege to do so. No rebuilding of indexes is necessary in this case. See [Section 5.2.2, “Server System Variables”](#), which describes the rules governing how to set this variable.

If you modify full-text variables that affect indexing (`ft_min_word_len`, `ft_max_word_len`, or `ft_stopword_file`), or if you change the stopword file itself, you must rebuild your FULLTEXT indexes after making the changes and restarting the server. To rebuild the indexes in this case, it is sufficient to do a QUICK repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Each table that contains any FULLTEXT index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table will cause the server to see the table as corrupt and in need of repair.

Note that if you use **myisamchk** to perform an operation that modifies table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or stopword file values used by the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values to **myisamchk** that you use for **mysqld**. For example, if you have set the minimum word length to 3, you can repair a table with **myisamchk** like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that **myisamchk** and the server use the same values for full-text parameters, place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3
```

```
[myisamchk]
```

```
ft_min_word_len=3
```

An alternative to using **myisamchk** is to use the REPAIR TABLE, ANALYZE TABLE, OPTIMIZE TABLE, or ALTER TABLE statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

12.8. Cast Functions and Operators

- **BINARY**

The **BINARY** operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column isn't defined as **BINARY** or **BLOB**. **BINARY** also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
      -> 1
mysql> SELECT BINARY 'a' = 'A';
      -> 0
mysql> SELECT 'a' = 'a ';
      -> 1
mysql> SELECT BINARY 'a' = 'a ';
      -> 0
```

In a comparison, **BINARY** affects the entire operation; it can be given before either operand with the same result.

BINARY str is shorthand for **CAST(str AS BINARY)**.

Note that in some contexts, if you cast an indexed column to **BINARY**, MySQL is not able to use the index efficiently.

- **CAST(expr AS type)**, **CONVERT(expr,type)**, **CONVERT(expr USING transcoding_name)**

The **CAST()** and **CONVERT()** functions take a value of one type and produce a value of another type.

The *type* can be one of the following values:

- **BINARY[(N)]**
- **CHAR[(N)]**
- **DATE**

- DATETIME
- DECIMAL
- SIGNED [INTEGER]
- TIME
- UNSIGNED [INTEGER]

BINARY produces a string with the BINARY data type. See [Section 11.4.2, “The BINARY and VARBINARY Types”](#) for a description of how this affects comparisons. If the optional length *N* is given, BINARY(*N*) causes the cast to use no more than *N* bytes of the argument. As of MySQL 5.0.17, values shorter than *N* bytes are padded with 0x00 bytes to a length of *N*.

CHAR(*N*) causes the cast to use no more than *N* characters of the argument.

The DECIMAL type is available as of MySQL 5.0.8.

CAST() and CONVERT(... USING ...) are standard SQL syntax. The non-USING form of CONVERT() is ODBC syntax.

CONVERT() with USING is used to convert data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string 'abc' in the default character set to the corresponding string in the utf8 character set:

```
SELECT CONVERT('abc' USING utf8);
```

Normally, you cannot compare a BLOB value or other binary string in case-insensitive fashion because binary strings have no character set, and thus no concept of lettercase. To perform a case-insensitive comparison, use the CONVERT() function to convert the value to a non-binary string. If the character set of the result has a case-insensitive collation, the LIKE operation is not case sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

To use a different character set, substitute its name for latin1 in the preceding

statement. To ensure that a case-insensitive collation is used, specify a COLLATE clause following the CONVERT() call.

CONVERT() can be used more generally for comparing strings that are represented in different character sets.

The cast functions are useful when you want to create a column with a specific type in a CREATE ... SELECT statement:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

The functions also can be useful for sorting ENUM columns in lexical order. Normally, sorting of ENUM columns occurs using the internal numeric values. Casting the values to CHAR results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

CAST(str AS BINARY) is the same thing as BINARY str. CAST(expr AS CHAR) treats the expression as a string with the default character set.

CAST() also changes the result if you use it as part of a more complex expression such as CONCAT('Date: ', CAST(NOW() AS DATE)).

You should not use CAST() to extract data in different formats but instead use string functions like LEFT() or EXTRACT(). See [Section 12.5, “Date and Time Functions”](#).

To cast a string to a numeric value in numeric context, you normally do not have to do anything other than to use the string value as though it were a number:

```
mysql> SELECT 1+'1';  
-> 2
```

If you use a number in string context, the number automatically is converted to a BINARY string.

```
mysql> SELECT CONCAT('hello you ',2);  
-> 'hello you 2'
```

MySQL supports arithmetic with both signed and unsigned 64-bit values. If you are using numeric operators (such as + or -) and one of the operands is an unsigned integer, the result is unsigned. You can override this by using the

SIGNED and UNSIGNED cast operators to cast the operation to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)  
-> 18446744073709551615  
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);  
-> -1
```

Note that if either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, DECIMAL column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;  
-> -1.0
```

If you are using a string in an arithmetic operation, this is converted to a floating-point number.

If you convert a “zero” date string to a date, CONVERT() and CAST() return NULL when the NO_ZERO_DATE SQL mode is enabled. As of MySQL 5.0.4, they also produce a warning.

12.9. Other Functions

12.9.1. Bit Functions

MySQL uses BIGINT (64-bit) arithmetic for bit operations, so these operators have a maximum range of 64 bits.

- |

Bitwise OR:

```
mysql> SELECT 29 | 15;
      -> 31
```

The result is an unsigned 64-bit integer.

- &

Bitwise AND:

```
mysql> SELECT 29 & 15;
      -> 13
```

The result is an unsigned 64-bit integer.

- ^

Bitwise XOR:

```
mysql> SELECT 1 ^ 1;
      -> 0
mysql> SELECT 1 ^ 0;
      -> 1
mysql> SELECT 11 ^ 3;
      -> 8
```

The result is an unsigned 64-bit integer.

- <<

Shifts a longlong (BIGINT) number to the left.

```
mysql> SELECT 1 << 2;
      -> 4
```

The result is an unsigned 64-bit integer.

- >>

Shifts a longlong (BIGINT) number to the right.

```
mysql> SELECT 4 >> 2;
      -> 1
```

The result is an unsigned 64-bit integer.

- ~

Invert all bits.

```
mysql> SELECT 5 & ~1;
      -> 4
```

The result is an unsigned 64-bit integer.

- BIT_COUNT(N)

Returns the number of bits that are set in the argument *N*.

```
mysql> SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
      -> 4, 3
```

12.9.2. Encryption and Compression Functions

The functions in this section perform encryption and decryption, and compression and uncompression:

Compression or encryption	Uncompression or decryption
AES_ENCRYPT()	AES_DECRYPT()
COMPRESS()	UNCOMPRESS()
ENCODE()	DECODE()
DES_ENCRYPT()	DES_DECRYPT()

ENCRYPT()	Not available
MD5()	Not available
OLD_PASSWORD()	Not available
PASSWORD()	Not available
SHA() or SHA1()	Not available
Not available	UNCOMPRESSED_LENGTH()

Note: The encryption and compression functions return binary strings. For many of these functions, the result might contain arbitrary byte values. If you want to store these results, use a BLOB column rather than a CHAR or (before MySQL 5.0.3) VARCHAR column to avoid potential problems with trailing space removal that would change data values.

Note: Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using one of the other encryption functions described in this section instead.

- `AES_ENCRYPT(str, key_str), AES_DECRYPT(encrypt_str, key_str)`

These functions allow encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” Encoding with a 128-bit key length is used, but you can extend it up to 256 bits by modifying the source. We chose 128 bits because it is much faster and it is secure enough for most purposes.

`AES_ENCRYPT()` encrypts a string and returns a binary string.
`AES_DECRYPT()` decrypts the encrypted string and returns the original string.
The input arguments may be any length. If either argument is NULL, the result of this function is also NULL.

Because AES is a block-level algorithm, padding is used to encode uneven length strings and so the result string length may be calculated using this formula:

$$16 \times (\text{trunc}(\text{string_length} / 16) + 1)$$

If `AES_DECRYPT()` detects invalid data or incorrect padding, it returns NULL. However, it is possible for `AES_DECRYPT()` to return a non-NULL value

(possibly garbage) if the input data or the key is invalid.

You can use the AES functions to store data in an encrypted form by modifying your queries:

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

AES_ENCRYPT() and AES_DECRYPT() can be considered the most cryptographically secure encryption functions currently available in MySQL.

- COMPRESS(string_to_compress)

Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`. The compressed string can be uncompressed with `UNCOMPRESS()`.

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
-> 21
mysql> SELECT LENGTH(COMPRESS(''));
-> 0
mysql> SELECT LENGTH(COMPRESS('a'));
-> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
-> 15
```

The compressed string contents are stored the following way:

- Empty strings are stored as empty strings.
 - Non-empty strings are stored as a four-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra `'.'` character is added to avoid problems with endspace trimming should the result be stored in a `CHAR` or `VARCHAR` column. (Use of `CHAR` or `VARCHAR` to store compressed strings is not recommended. It is better to use a `BLOB` column instead.)
- DECODE(crypt_str,pass_str)

Decrypts the encrypted string *crypt_str* using *pass_str* as the password.

crypt_str should be a string returned from `ENCODE()`.

- `ENCODE(str,pass_str)`

Encrypt *str* using *pass_str* as the password. To decrypt the result, use `DECODE()`.

The result is a binary string of the same length as *str*.

The strength of the encryption is based on how good the random generator is. It should suffice for short strings.

- `DES_DECRYPT(crypt_str[,key_str])`

Decrypts a string encrypted with `DES_ENCRYPT()`. If an error occurs, this function returns `NULL`.

Note that this function works only if MySQL has been configured with SSL support. See [Section 5.9.7, “Using Secure Connections”](#).

If no *key_str* argument is given, `DES_DECRYPT()` examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the `SUPER` privilege. The key file can be specified with the `--des-key-file` server option.

If you pass this function a *key_str* argument, that string is used as the key for decrypting the message.

If the *crypt_str* argument does not appear to be an encrypted string, MySQL returns the given *crypt_str*.

- `DES_ENCRYPT(str[, {key_num|key_str}])`

Encrypts the string with the given key using the Triple-DES algorithm.

Note that this function works only if MySQL has been configured with SSL support. See [Section 5.9.7, “Using Secure Connections”](#).

The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()`, if one was given:

Argument	Description
No argument	The first key from the DES key file is used.
<i>key_num</i>	The given key number (0-9) from the DES key file is used.
<i>key_str</i>	The given key string is used to encrypt <i>str</i> .

The key file can be specified with the `--des-key-file` server option.

The return string is a binary string where the first character is `CHAR(128 | key_num)`. If an error occurs, `DES_ENCRYPT()` returns `NULL`.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *key_num* is 127.

The string length for the result is given by this formula:

$$new_len = orig_len + (8 - (orig_len \% 8)) + 1$$

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each *key_num* value must be a number in the range from 0 to 9. Lines in the file may be in any order. *des_key_str* is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()`.

You can tell MySQL to read new key values from the key file with the `FLUSH DES_KEY_FILE` statement. This requires the `RELOAD` privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table  
      > WHERE crypted_credit_card = DES_ENCRYPT('credit_card_numk
```

- `ENCRYPT(str[,salt])`

Encrypts *str* using the Unix `crypt()` system call and returns a binary string. The *salt* argument should be a string with at least two characters. If no *salt* argument is given, a random value is used.

```
mysql> SELECT ENCRYPT('hello');
        -> 'VxuFAJXVARROc'
```

`ENCRYPT()` ignores all but the first eight characters of *str*, at least on some systems. This behavior is determined by the implementation of the underlying `crypt()` system call.

If `crypt()` is not available on your system (as is the case with Windows), `ENCRYPT()` always returns `NULL`.

- `MD5(str)`

Calculates an MD5 128-bit checksum for the string. The value is returned as a binary string of 32 hex digits, or `NULL` if the argument was `NULL`. The return value can, for example, be used as a hash key.

```
mysql> SELECT MD5('testing');
        -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

This is the “RSA Data Security, Inc. MD5 Message-Digest Algorithm.”

If you want to convert the value to uppercase, see the description of binary string conversion given in the entry for the `BINARY` operator in [Section 12.8, “Cast Functions and Operators”](#).

See the note regarding the MD5 algorithm at the beginning this section.

- `OLD_PASSWORD(str)`

`OLD_PASSWORD()` was added to MySQL when the implementation of `PASSWORD()` was changed to improve security. `OLD_PASSWORD()` returns the value of the old (pre-4.1) implementation of `PASSWORD()` as a binary string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to your version 5.0 MySQL server without locking them out. See [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#).

- PASSWORD(str)

Calculates and returns a password string from the plaintext password *str* and returns a binary string, or NULL if the argument was NULL. This is the function that is used for encrypting MySQL passwords for storage in the Password column of the user grant table.

```
mysql> SELECT PASSWORD('badpwd');  
-> '*AAB3E285149C0135D51A520E1940DD3263DC008C'
```

PASSWORD() encryption is one-way (not reversible).

PASSWORD() does not perform password encryption in the same way that Unix passwords are encrypted. See ENCRYPT().

Note: The PASSWORD() function is used by the authentication system in MySQL Server; you should *not* use it in your own applications. For that purpose, consider MD5() or SHA1() instead. Also see [RFC 2195, section 2 \(Challenge-Response Authentication Mechanism \(CRAM\)\)](#), for more information about handling passwords and authentication securely in your applications.

- SHA1(str), SHA(str)

Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a binary string of 40 hex digits, or NULL if the argument was NULL. One of the possible uses for this function is as a hash key. You can also use it as a cryptographic function for storing passwords. SHA() is synonymous with SHA1().

```
mysql> SELECT SHA1('abc');  
-> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

SHA1() can be considered a cryptographically more secure equivalent of MD5(). However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

- UNCOMPRESS(string_to_uncompress)

Uncompresses a string compressed by the COMPRESS() function. If the argument is not a compressed value, the result is NULL. This function

requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
        -> 'any string'
mysql> SELECT UNCOMPRESS('any string');
        -> NULL
```

- `UNCOMPRESSED_LENGTH(compressed_string)`

Returns the length that the compressed string had before being compressed.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a', 30)));
        -> 30
```

12.9.3. Information Functions

- `BENCHMARK(count,expr)`

The `BENCHMARK()` function executes the expression `expr` repeatedly `count` times. It may be used to time how quickly MySQL processes the expression. The result value is always `0`. The intended use is from within the **mysql** client, which reports query execution times:

```
mysql> SELECT BENCHMARK(1000000, ENCODE('hello', 'goodbye'));
+-----+
| BENCHMARK(1000000, ENCODE('hello', 'goodbye')) |
+-----+
|                                                    0 |
+-----+
1 row in set (4.74 sec)
```

The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute `BENCHMARK()` several times, and to interpret the result with regard to how heavily loaded the server machine is.

- `CHARSET(str)`

Returns the character set of the string argument.

```
mysql> SELECT CHARSET('abc');
        -> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
        -> 'utf8'
```

```
mysql> SELECT CHARSET(USER());
      -> 'utf8'
```

- COERCIBILITY(str)

Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
      -> 0
mysql> SELECT COERCIBILITY(USER());
      -> 3
mysql> SELECT COERCIBILITY('abc');
```

The return values have the meanings shown in the following table. Lower values have higher precedence.

Coercibility	Meaning	Example
0	Explicit collation	Value with COLLATE clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value
3	System constant	USER() return value
4	Coercible	Literal string
5	Ignorable	NULL or an expression derived from NULL

Before MySQL 5.0.3, the return values are shown as follows, and functions such as USER() have a coercibility of 2:

Coercibility	Meaning	Example
0	Explicit collation	Value with COLLATE clause
1	No collation	Concatenation of strings with different collations
2	Implicit	Column value, stored routine parameter or

	collation	local variable
3	Coercible	Literal string

- **COLLATION(str)**

Returns the collation of the string argument.

```
mysql> SELECT COLLATION('abc');
      -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
      -> 'utf8_general_ci'
```

- **CONNECTION_ID()**

Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

```
mysql> SELECT CONNECTION_ID();
      -> 23786
```

- **CURRENT_USER, CURRENT_USER()**

Returns the username and hostname combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges. As of MySQL 5.0.10, within a stored routine that is defined with the SQL SECURITY DEFINER characteristic, **CURRENT_USER()** returns the creator of the routine. The return value is a string in the `utf8` character set.

The value of **CURRENT_USER()** can differ from the value of **USER()**.

```
mysql> SELECT USER();
      -> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
      -> '@localhost'
```

The example illustrates that although the client specified a username of `davida` (as indicated by the value of the **USER()** function), the server authenticated the client using an anonymous user account (as seen by the

empty username part of the `CURRENT_USER()` value). One way this might occur is that there is no account listed in the grant tables for `davida`.

- `DATABASE()`

Returns the default (current) database name as a string in the utf8 character set. If there is no default database, `DATABASE()` returns `NULL`. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

```
mysql> SELECT DATABASE();  
      -> 'test'
```

- `FOUND_ROWS()`

A `SELECT` statement may include a `LIMIT` clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the `LIMIT`, but without running the statement again. To obtain this row count, include a `SQL_CALC_FOUND_ROWS` option in the `SELECT` statement, and then invoke `FOUND_ROWS()` afterward:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name  
      -> WHERE id > 100 LIMIT 10;  
mysql> SELECT FOUND_ROWS();
```

The second `SELECT` returns a number indicating how many rows the first `SELECT` would have returned had it been written without the `LIMIT` clause. (If the preceding `SELECT` statement does not include the `SQL_CALC_FOUND_ROWS` option, then `FOUND_ROWS()` may return a different result when `LIMIT` is used than when it is not.)

The row count available through `FOUND_ROWS()` is transient and not intended to be available past the statement following the `SELECT SQL_CALC_FOUND_ROWS` statement. If you need to refer to the value later, save it:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;  
mysql> SET @rows = FOUND_ROWS();
```

If you are using `SELECT SQL_CALC_FOUND_ROWS`, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without `LIMIT`, because the result set need not be sent to the client.

`SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display containing links to the pages that show other sections of a search result. Using `FOUND_ROWS()` allows you to determine how many other pages are needed for the rest of the result.

The use of `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` is more complex for `UNION` statements than for simple `SELECT` statements, because `LIMIT` may occur at multiple places in a `UNION`. It may be applied to individual `SELECT` statements in the `UNION`, or global to the `UNION` result as a whole.

The intent of `SQL_CALC_FOUND_ROWS` for `UNION` is that it should return the row count that would be returned without a global `LIMIT`. The conditions for use of `SQL_CALC_FOUND_ROWS` with `UNION` are:

- The `SQL_CALC_FOUND_ROWS` keyword must appear in the first `SELECT` of the `UNION`.
 - The value of `FOUND_ROWS()` is exact only if `UNION ALL` is used. If `UNION` without `ALL` is used, duplicate removal occurs and the value of `FOUND_ROWS()` is only approximate.
 - If no `LIMIT` is present in the `UNION`, `SQL_CALC_FOUND_ROWS` is ignored and returns the number of rows in the temporary table that is created to process the `UNION`.
- `LAST_INSERT_ID()`, `LAST_INSERT_ID(expr)`

Returns the *first* automatically generated value that was set for an `AUTO_INCREMENT` column by the *most recent* `INSERT` or `UPDATE` statement to affect such a column.

```
mysql> SELECT LAST_INSERT_ID();
```

-> 195

The ID that was generated is maintained in the server on a *per-connection basis*. This means that the value returned by the function to a given client is the first `AUTO_INCREMENT` value generated for most recent statement affecting an `AUTO_INCREMENT` column *by that client*. This value cannot be affected by other clients, even if they generate `AUTO_INCREMENT` values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

The value of `LAST_INSERT_ID()` is not changed if you set the `AUTO_INCREMENT` column of a row to a non-“magic” value (that is, a value that is not `NULL` and not `0`).

Important: If you insert multiple rows using a single `INSERT` statement, `LAST_INSERT_ID()` returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same `INSERT` statement against some other server.

For example:

```
mysql> USE test;
Database changed
mysql> CREATE TABLE t (
  ->   id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  ->   name VARCHAR(10) NOT NULL
  -> );
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> INSERT INTO t VALUES (NULL, 'Bob');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> SELECT * FROM t;
+-----+-----+
| id | name |
+-----+-----+
|  1 | Bob  |
+-----+-----+
1 row in set (0.01 sec)
```

```
mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
```

```
|          1 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> INSERT INTO t VALUES
-> (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM t;
+----+-----+
| id | name |
+----+-----+
|  1 | Bob  |
|  2 | Mary |
|  3 | Jane |
|  4 | Lisa |
+----+-----+
4 rows in set (0.01 sec)
```

```
mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          2 |
+-----+
1 row in set (0.00 sec)
```

Although the second `INSERT` statement inserted three new rows into `t`, the ID generated for the first of these rows was 2, and it is this value that is returned by `LAST_INSERT_ID()` for the following `SELECT` statement.

If you use `INSERT IGNORE` and the row is ignored, the `AUTO_INCREMENT` counter is not incremented and `LAST_INSERT_ID()` returns 0, which reflects that no row was inserted.

If *expr* is given as an argument to `LAST_INSERT_ID()`, the value of the argument is returned by the function and is remembered as the next value to be returned by `LAST_INSERT_ID()`. This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

2. Use the table to generate sequence numbers like this:

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

The UPDATE statement increments the sequence counter and causes the next call to LAST_INSERT_ID() to return the updated value. The SELECT statement retrieves that value. The mysql_insert_id() C API function can also be used to get the value. See [Section 22.2.3.36](#), “[mysql_insert_id\(\)](#)”.

You can generate sequences without calling LAST_INSERT_ID(), but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the UPDATE statement and get their own sequence value with the SELECT statement (or mysql_insert_id()), without affecting or being affected by other clients that generate their own sequence values.

Note that mysql_insert_id() is only updated after INSERT and UPDATE statements, so you cannot use the C API function to retrieve the value for LAST_INSERT_ID(expr) after executing other SQL statements like SELECT or SET.

- ROW_COUNT()

ROW_COUNT() returns the number of rows updated, inserted, or deleted by the preceding statement. This is the same as the row count that the **mysql** client displays and the value from the mysql_affected_rows() C API function.

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|           3 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)
```

```
mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|          2 |
+-----+
1 row in set (0.00 sec)
```

ROW_COUNT() was added in MySQL 5.0.1.

- SCHEMA()

This function is a synonym for DATABASE(). It was added in MySQL 5.0.2.

- SESSION_USER()

SESSION_USER() is a synonym for USER().

- SYSTEM_USER()

SYSTEM_USER() is a synonym for USER().

- USER()

Returns the current MySQL username and hostname as a string in the utf8 character set.

```
mysql> SELECT USER();
-> 'david@localhost'
```

The value indicates the username you specified when connecting to the server, and the client host from which you connected. The value can be different from that of CURRENT_USER().

You can extract only the username part like this:

```
mysql> SELECT SUBSTRING_INDEX(USER(), '@', 1);
-> 'david'
```

- VERSION()

Returns a string that indicates the MySQL server version. The string uses

the utf8 character set.

```
mysql> SELECT VERSION();  
      -> '5.0.25-standard'
```

Note that if your version string ends with `-log` this means that logging is enabled.

12.9.4. Miscellaneous Functions

- `DEFAULT(col_name)`

Returns the default value for a table column. Starting with MySQL 5.0.2, an error results if the column has no default value.

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

- `FORMAT(X,D)`

Formats the number *X* to a format like `'#,###,###.##'`, rounded to *D* decimal places, and returns the result as a string. For details, see [Section 12.3, “String Functions”](#).

- `GET_LOCK(str,timeout)`

Tries to obtain a lock with a name given by the string *str*, using a timeout of *timeout* seconds. Returns 1 if the lock was obtained successfully, 0 if the attempt timed out (for example, because another client has previously locked the name), or NULL if an error occurred (such as running out of memory or the thread was killed with **mysqladmin kill**). If you have a lock obtained with `GET_LOCK()`, it is released when you execute `RELEASE_LOCK()`, execute a new `GET_LOCK()`, or your connection terminates (either normally or abnormally). Locks obtained with `GET_LOCK()` do not interact with transactions. That is, committing a transaction does not release any such locks obtained during the transaction.

This function can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked by one client, `GET_LOCK()` blocks any request by another client for a lock with the same name. This allows clients that agree on a given lock

name to use the name to perform cooperative advisory locking. But be aware that it also allows a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form *db_name.str* or *app_name.str*.

```
mysql> SELECT GET_LOCK('lock1',10);
-> 1
mysql> SELECT IS_FREE_LOCK('lock2');
-> 1
mysql> SELECT GET_LOCK('lock2',10);
-> 1
mysql> SELECT RELEASE_LOCK('lock2');
-> 1
mysql> SELECT RELEASE_LOCK('lock1');
-> NULL
```

The second `RELEASE_LOCK()` call returns `NULL` because the lock `'lock1'` was automatically released by the second `GET_LOCK()` call.

Note: If a client attempts to acquire a lock that is already held by another client, it blocks according to the *timeout* argument. If the blocked client terminates, its thread does not die until the lock request times out. This is a known bug.

- `INET_ATON(expr)`

Given the dotted-quad representation of a network address as a string, returns an integer that represents the numeric value of the address. Addresses may be 4- or 8-byte addresses.

```
mysql> SELECT INET_ATON('209.207.224.40');
-> 3520061480
```

The generated number is always in network byte order. For the example just shown, the number is calculated as $209 \times 256^3 + 207 \times 256^2 + 224 \times 256 + 40$.

`INET_ATON()` also understands short-form IP addresses:

```
mysql> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');
-> 2130706433, 2130706433
```

Note: When storing values generated by `INET_ATON()`, it is recommended that you use an `INT UNSIGNED` column. If you use a (signed) `INT` column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See [Section 11.2, “Numeric Types”](#).

- `INET_NTOA(expr)`

Given a numeric network address (4 or 8 byte), returns the dotted-quad representation of the address as a string.

```
mysql> SELECT INET_NTOA(3520061480);  
-> '209.207.224.40'
```

- `IS_FREE_LOCK(str)`

Checks whether the lock named `str` is free to use (that is, not locked). Returns 1 if the lock is free (no one is using the lock), 0 if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument).

- `IS_USED_LOCK(str)`

Checks whether the lock named `str` is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns `NULL`.

- `MASTER_POS_WAIT(log_name,log_pos[,timeout])`

This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events the slave had to wait for to advance to the specified position. The function returns `NULL` if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns -1 if the timeout has been exceeded. If the slave SQL thread stops while `MASTER_POS_WAIT()` is waiting, the function returns `NULL`. If the slave is past the specified position, the function returns immediately.

If a `timeout` value is specified, `MASTER_POS_WAIT()` stops waiting when `timeout` seconds have elapsed. `timeout` must be greater than 0; a zero or negative `timeout` means no timeout.

- `NAME_CONST(name,value)`

Returns the given value. When used to produce a result set column, `NAME_CONST()` causes the column to have the given name.

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|      14 |
+-----+
```

This function was added in MySQL 5.0.12. It is for internal use only. The server uses it when writing statements from stored routines that contain references to local routine variables, as described in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#). You might see this function in the output from `mysqlbinlog`.

- `RELEASE_LOCK(str)`

Releases the lock named by the string *str* that was obtained with `GET_LOCK()`. Returns 1 if the lock was released, 0 if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` or if it has previously been released.

The `DO` statement is convenient to use with `RELEASE_LOCK()`. See [Section 13.2.2, “DO Syntax”](#).

- `SLEEP(duration)`

Sleeps (pauses) for the number of seconds given by the *duration* argument, then returns 0. If `SLEEP()` is interrupted, it returns 1. The duration may have a fractional part given in microseconds. This function was added in MySQL 5.0.12.

- `UUID()`

Returns a Universal Unique Identifier (UUID) generated according to “DCE 1.1: Remote Procedure Call” (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in

October 1997 (Document Number C706,
<http://www.opengroup.org/public/pubs/catalog/c706.htm>).

A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

A UUID is a 128-bit number represented by a string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

- The first three numbers are generated from a timestamp.
- The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
- The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have *very* low probability.

Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MySQL uses a randomly generated 48-bit number.

```
mysql> SELECT UUID();  
-> '6ccd780c-baba-1026-9564-0040f4311e29'
```

Note that `UUID()` does not yet work with replication.

- `VALUES(col_name)`

In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that

would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in `INSERT ... ON DUPLICATE KEY UPDATE` statements and returns `NULL` otherwise. [Section 13.2.4.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
-> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

12.10. Functions and Modifiers for Use with GROUP BY Clauses

12.10.1. GROUP BY (Aggregate) Functions

This section describes group (aggregate) functions that operate on sets of values. Unless otherwise stated, group functions ignore NULL values.

If you use a group function in a statement containing no GROUP BY clause, it is equivalent to grouping on all rows.

The SUM() and AVG() aggregate functions do not work with temporal values. (They convert the values to numbers, which loses the part after the first non-numeric character.) To work around this problem, you can convert to numeric units, perform the aggregate operation, and convert back to a temporal value.

Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;  
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

- AVG([DISTINCT] *expr*)

Returns the average value of *expr*. The DISTINCT option can be used as of MySQL 5.0.3 to return the average of the distinct values of *expr*.

AVG() returns NULL if there were no matching rows.

```
mysql> SELECT student_name, AVG(test_score)  
-> FROM student  
-> GROUP BY student_name;
```

- BIT_AND(*expr*)

Returns the bitwise AND of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision.

This function returns 18446744073709551615 if there were no matching rows. (This is the value of an unsigned BIGINT value with all bits set to 1.)

- `BIT_OR(expr)`

Returns the bitwise OR of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision.

This function returns 0 if there were no matching rows.

- `BIT_XOR(expr)`

Returns the bitwise XOR of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision.

This function returns 0 if there were no matching rows.

- `COUNT(expr)`

Returns a count of the number of non-NULL values in the rows retrieved by a SELECT statement. The result is a BIGINT value.

`COUNT()` returns 0 if there were no matching rows.

```
mysql> SELECT student.student_name,COUNT(*)
->      FROM student,course
->      WHERE student.student_id=course.student_id
->      GROUP BY student_name;
```

`COUNT(*)` is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain NULL values.

`COUNT(*)` is optimized to return very quickly if the SELECT retrieves from one table, no other columns are retrieved, and there is no WHERE clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```

This optimization applies only to MyISAM tables only, because an exact row count is stored for this storage engine and can be accessed very quickly. For transactional storage engines such as InnoDB and BDB, storing an exact row count is more problematic because multiple transactions may be occurring, each of which may affect the count.

- `COUNT(DISTINCT expr,[expr...])`

Returns a count of the number of different non-NULL values.

COUNT(DISTINCT) returns 0 if there were no matching rows.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

In MySQL, you can obtain the number of distinct expression combinations that do not contain NULL by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside COUNT(DISTINCT ...).

- GROUP_CONCAT(expr)

This function returns a string result with the concatenated non-NULL values from a group. It returns NULL if there are no non-NULL values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
              [ORDER BY {unsigned_integer | col_name | expr}
               [ASC | DESC] [,col_name ...]]
              [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
->        GROUP_CONCAT(test_score)
->        FROM student
->        GROUP BY student_name;
```

Or:

```
mysql> SELECT student_name,
->        GROUP_CONCAT(DISTINCT test_score
->                      ORDER BY test_score DESC SEPARATOR ' ')
->        FROM student
->        GROUP BY student_name;
```

In MySQL, you can get the concatenated values of expression combinations. You can eliminate duplicate values by using DISTINCT. If you want to sort values in the result, you should use ORDER BY clause. To sort in reverse order, add the DESC (descending) keyword to the name of the column you are sorting by in the ORDER BY clause. The default is ascending order; this may be specified explicitly using the ASC keyword. SEPARATOR is followed by the string value that should be inserted between values of result. The default is a comma (','). You can eliminate the separator

altogether by specifying `SEPARATOR ' '`.

You can set a maximum allowed length with the `group_concat_max_len` system variable. (The default value is 1024.) The syntax to do this at runtime is as follows, where *val* is an unsigned integer:

```
SET [SESSION | GLOBAL] group_concat_max_len = val;
```

If a maximum length has been set, the result is truncated to this maximum length.

Beginning with MySQL 5.0.19, the type returned by `GROUP_CONCAT()` is always `VARCHAR` unless `group_concat_max_len` is greater than 512, in which case, it returns a `BLOB`. (Previously, it returned a `BLOB` with `group_concat_max_len` greater than 512 only if the query included an `ORDER BY` clause.)

See also `CONCAT()` and `CONCAT_WS()`: [Section 12.3, “String Functions”](#).

- `MIN([DISTINCT] expr), MAX([DISTINCT] expr)`

Returns the minimum or maximum value of *expr*. `MIN()` and `MAX()` may take a string argument; in such cases they return the minimum or maximum string value. See [Section 7.4.5, “How MySQL Uses Indexes”](#). The `DISTINCT` keyword can be used to find the minimum or maximum of the distinct values of *expr*, however, this produces the same result as omitting `DISTINCT`.

`MIN()` and `MAX()` return `NULL` if there were no matching rows.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
->      FROM student
->      GROUP BY student_name;
```

For `MIN()`, `MAX()`, and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set. This differs from how `ORDER BY` compares them. This is expected to be rectified in a future MySQL release.

- `STD(expr) STDDEV(expr)`

Returns the population standard deviation of *expr*. This is an extension to standard SQL. The `STDDEV()` form of this function is provided for compatibility with Oracle. As of MySQL 5.0.3, the standard SQL function `STDDEV_POP()` can be used instead.

These functions return `NULL` if there were no matching rows.

- `STDDEV_POP(expr)`

Returns the population standard deviation of *expr* (the square root of `VAR_POP()`). This function was added in MySQL 5.0.3. Before 5.0.3, you can use `STD()` or `STDDEV()`, which are equivalent but not standard SQL.

`STDDEV_POP()` returns `NULL` if there were no matching rows.

- `STDDEV_SAMP(expr)`

Returns the sample standard deviation of *expr* (the square root of `VAR_SAMP()`). This function was added in MySQL 5.0.3.

`STDDEV_SAMP()` returns `NULL` if there were no matching rows.

- `SUM([DISTINCT] expr)`

Returns the sum of *expr*. If the return set has no rows, `SUM()` returns `NULL`. The `DISTINCT` keyword can be used in MySQL 5.0 to sum only the distinct values of *expr*.

`SUM()` returns `NULL` if there were no matching rows.

- `VAR_POP(expr)`

Returns the population standard variance of *expr*. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. This function was added in MySQL 5.0.3. Before 5.0.3, you can use `VARIANCE()`, which is equivalent but is not standard SQL.

`VAR_POP()` returns `NULL` if there were no matching rows.

- `VAR_SAMP(expr)`

Returns the sample variance of *expr*. That is, the denominator is the number of rows minus one. This function was added in MySQL 5.0.3.

`VAR_SAMP()` returns NULL if there were no matching rows.

- `VARIANCE(expr)`

Returns the population standard variance of *expr*. This is an extension to standard SQL. As of MySQL 5.0.3, the standard SQL function `VAR_POP()` can be used instead.

`VARIANCE()` returns NULL if there were no matching rows.

12.10.2. GROUP BY Modifiers

The `GROUP BY` clause allows a `WITH ROLLUP` modifier that causes extra rows to be added to the summary output. These rows represent higher-level (or super-aggregate) summary operations. `ROLLUP` thus allows you to answer questions at multiple levels of analysis with a single query. It can be used, for example, to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a table named `sales` has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales
(
  year      INT NOT NULL,
  country   VARCHAR(20) NOT NULL,
  product   VARCHAR(32) NOT NULL,
  profit    INT
);
```

The table's contents can be summarized per year with a simple `GROUP BY` like this:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+-----+-----+
| year | SUM(profit) |
+-----+-----+
| 2000 |          4525 |
| 2001 |          3010 |
+-----+-----+
```

This output shows the total profit for each year, but if you also want to determine the total profit summed over all years, you must add up the individual values yourself or run an additional query.

Or you can use `ROLLUP`, which provides both levels of analysis with a single query. Adding a `WITH ROLLUP` modifier to the `GROUP BY` clause causes the query to produce another row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP
+-----+-----+
| year | SUM(profit) |
+-----+-----+
| 2000 |          4525 |
| 2001 |          3010 |
| NULL |          7535 |
+-----+-----+
```

The grand total super-aggregate line is identified by the value `NULL` in the year column.

`ROLLUP` has a more complex effect when there are multiple `GROUP BY` columns. In this case, each time there is a “break” (change in value) in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without `ROLLUP`, a summary on the sales table based on year, country, and product might look like this:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product;
+-----+-----+-----+-----+
| year | country | product | SUM(profit) |
+-----+-----+-----+-----+
| 2000 | Finland | Computer |          1500 |
| 2000 | Finland | Phone    |           100 |
| 2000 | India   | Calculator |           150 |
| 2000 | India   | Computer |          1200 |
| 2000 | USA     | Calculator |            75 |
| 2000 | USA     | Computer |          1500 |
| 2001 | Finland | Phone    |            10 |
| 2001 | USA     | Calculator |            50 |
| 2001 | USA     | Computer |          2700 |
| 2001 | USA     | TV       |           250 |
+-----+-----+-----+-----+
```

The output indicates summary values only at the year/country/product level of analysis. When ROLLUP is added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	Finland	NULL	1600
2000	India	Calculator	150
2000	India	Computer	1200
2000	India	NULL	1350
2000	USA	Calculator	75
2000	USA	Computer	1500
2000	USA	NULL	1575
2000	NULL	NULL	4525
2001	Finland	Phone	10
2001	Finland	NULL	10
2001	USA	Calculator	50
2001	USA	Computer	2700
2001	USA	TV	250
2001	USA	NULL	3000
2001	NULL	NULL	3010
NULL	NULL	NULL	7535

For this query, adding ROLLUP causes the output to include summary information at four levels of analysis, not just one. Here's how to interpret the ROLLUP output:

- Following each set of product rows for a given year and country, an extra summary row is produced showing the total for all products. These rows have the product column set to NULL.
- Following each set of rows for a given year, an extra summary row is produced showing the total for all countries and products. These rows have the country and products columns set to NULL.
- Finally, following all other rows, an extra summary row is produced showing the grand total for all years, countries, and products. This row has the year, country, and products columns set to NULL.

Other Considerations When using ROLLUP

The following items list some behaviors specific to the MySQL implementation of ROLLUP:

When you use ROLLUP, you cannot also use an ORDER BY clause to sort the results. In other words, ROLLUP and ORDER BY are mutually exclusive. However, you still have some control over sort order. GROUP BY in MySQL sorts results, and you can use explicit ASC and DESC keywords with columns named in the GROUP BY list to specify sort order for individual columns. (The higher-level summary rows added by ROLLUP still appear after the rows from which they are calculated, regardless of the sort order.)

LIMIT can be used to restrict the number of rows returned to the client. LIMIT is applied after ROLLUP, so the limit applies against the extra rows added by ROLLUP. For example:

```
mysql> SELECT year, country, product, SUM(profit)
-> FROM sales
-> GROUP BY year, country, product WITH ROLLUP
-> LIMIT 5;
```

year	country	product	SUM(profit)
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	Finland	NULL	1600
2000	India	Calculator	150
2000	India	Computer	1200

Using LIMIT with ROLLUP may produce results that are more difficult to interpret, because you have less context for understanding the super-aggregate rows.

The NULL indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the GROUP BY clause following the leftmost one that has changed value. For any column in the result set with a name that is a lexical match to any of those names, its value is set to NULL. (If you specify grouping columns by column number, the server identifies which columns to set to NULL by number.)

Because the NULL values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you cannot test them as NULL values within the query itself. For example, you cannot add HAVING product IS NULL

to the query to eliminate from the output all but the super-aggregate rows.

On the other hand, the NULL values do appear as NULL on the client side and can be tested as such using any MySQL client programming interface.

12.10.3. GROUP BY and HAVING with Hidden Fields

MySQL extends the use of GROUP BY so that you can use non-aggregated columns or calculations in the SELECT list that do not appear in the GROUP BY clause. You can use this feature to get better performance by avoiding unnecessary column sorting and grouping. For example, you do not need to group on customer.name in the following query:

```
SELECT order.custid, customer.name, MAX(payments)
FROM order, customer
WHERE order.custid = customer.custid
GROUP BY order.custid;
```

In standard SQL, you would have to add customer.name to the GROUP BY clause. In MySQL, the name is redundant.

Do *not* use this feature if the columns you omit from the GROUP BY part are not constant in the group. The server is free to return any value from the group, so the results are indeterminate unless all values are the same.

A similar MySQL extension applies to the HAVING clause. The SQL standard does not allow the HAVING clause to name any column that is not found in the GROUP BY clause if it is not enclosed in an aggregate function. MySQL allows the use of such columns to simplify calculations. This extension assumes that the non-grouped columns will have the same group-wise values. Otherwise, the result is indeterminate.

If the ONLY_FULL_GROUP_BY SQL mode is enabled, the MySQL extension to GROUP BY does not apply. That is, columns not named in the GROUP BY clause cannot be used in the SELECT list or HAVING clause if not used in an aggregate function.

The select list extension also applies to ORDER BY. That is, you can use non-aggregated columns or calculations in the ORDER BY clause that do not appear in the GROUP BY clause. This extension does not apply if the ONLY_FULL_GROUP_BY

SQL mode is enabled.

In some cases, you can use `MIN()` and `MAX()` to obtain a specific column value even if it isn't unique. The following gives the value of `column` from the row containing the smallest value in the `sort` column:

```
SUBSTR(MIN(CONCAT(RPAD(sort,6,' '),column)),7)
```

See [Section 3.6.4, “The Rows Holding the Group-wise Maximum of a Certain Field”](#).

Note that if you are trying to follow standard SQL, you can't use expressions in `GROUP BY` clauses. You can work around this limitation by using an alias for the expression:

```
SELECT id,FLOOR(value/100) AS val
FROM tbl_name
GROUP BY id, val;
```

MySQL does allow expressions in `GROUP BY` clauses. For example:

```
SELECT id,FLOOR(value/100)
FROM tbl_name
GROUP BY id, FLOOR(value/100);
```

Chapter 13. SQL Statement Syntax

Table of Contents

[13.1. Data Definition Statements](#)

[13.1.1. ALTER DATABASE Syntax](#)

[13.1.2. ALTER TABLE Syntax](#)

[13.1.3. CREATE DATABASE Syntax](#)

[13.1.4. CREATE INDEX Syntax](#)

[13.1.5. CREATE TABLE Syntax](#)

[13.1.6. DROP DATABASE Syntax](#)

[13.1.7. DROP INDEX Syntax](#)

[13.1.8. DROP TABLE Syntax](#)

[13.1.9. RENAME TABLE Syntax](#)

[13.2. Data Manipulation Statements](#)

[13.2.1. DELETE Syntax](#)

[13.2.2. DO Syntax](#)

[13.2.3. HANDLER Syntax](#)

[13.2.4. INSERT Syntax](#)

[13.2.5. LOAD DATA INFILE Syntax](#)

[13.2.6. REPLACE Syntax](#)

[13.2.7. SELECT Syntax](#)

[13.2.8. Subquery Syntax](#)

[13.2.9. TRUNCATE Syntax](#)

[13.2.10. UPDATE Syntax](#)

[13.3. MySQL Utility Statements](#)

[13.3.1. DESCRIBE Syntax](#)

[13.3.2. HELP Syntax](#)

[13.3.3. USE Syntax](#)

[13.4. MySQL Transactional and Locking Statements](#)

[13.4.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax](#)

[13.4.2. Statements That Cannot Be Rolled Back](#)

[13.4.3. Statements That Cause an Implicit Commit](#)

[13.4.4. SAVEPOINT and ROLLBACK TO SAVEPOINT Syntax](#)

[13.4.5. LOCK TABLES and UNLOCK TABLES Syntax](#)

[13.4.6. SET TRANSACTION Syntax](#)

[13.4.7. XA Transactions](#)

[13.5. Database Administration Statements](#)

[13.5.1. Account Management Statements](#)

[13.5.2. Table Maintenance Statements](#)

[13.5.3. SET Syntax](#)

[13.5.4. SHOW Syntax](#)

[13.5.5. Other Administrative Statements](#)

[13.6. Replication Statements](#)

[13.6.1. SQL Statements for Controlling Master Servers](#)

[13.6.2. SQL Statements for Controlling Slave Servers](#)

[13.7. SQL Syntax for Prepared Statements](#)

This chapter describes the syntax for most of the SQL statements supported by MySQL. Additional statement descriptions can be found in the following chapters:

- The EXPLAIN statement is discussed in [Chapter 7, Optimization](#).
- Statements for writing stored routines are covered in [Chapter 17, Stored Procedures and Functions](#).
- Statements for writing triggers are covered in [Chapter 18, Triggers](#).
- View-related statements are covered in [Chapter 19, Views](#).

13.1. Data Definition Statements

13.1.1. ALTER DATABASE Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification [, alter_specification] ...
```

alter_specification:

```
[DEFAULT] CHARACTER SET charset_name
| [DEFAULT] COLLATE collation_name
```

ALTER DATABASE enables you to change the overall characteristics of a database. These characteristics are stored in the `db.opt` file in the database directory. To use ALTER DATABASE, you need the ALTER privilege on the database. ALTER SCHEMA is a synonym for ALTER DATABASE as of MySQL 5.0.2.

The CHARACTER SET clause changes the default database character set. The COLLATE clause changes the default database collation. [Chapter 10, Character Set Support](#), discusses character set and collation names.

The database name can be omitted, in which case the statement applies to the default database.

13.1.2. ALTER TABLE Syntax

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...
```

alter_specification:

```
ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
| ADD [COLUMN] (column_definition,...)
| ADD {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
| ADD [CONSTRAINT [symbol]]
    PRIMARY KEY [index_type] (index_col_name,...)
| ADD [CONSTRAINT [symbol]]
    UNIQUE [INDEX|KEY] [index_name] [index_type] (index_col_name
| ADD [FULLTEXT|SPATIAL] [INDEX|KEY] [index_name] (index_col_name,
| ADD [CONSTRAINT [symbol]]
    FOREIGN KEY [index_name] (index_col_name,...)
    [reference_definition]
| ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
| CHANGE [COLUMN] old_col_name column_definition
```

```

        [FIRST|AFTER col_name]
| MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP {INDEX|KEY} index_name
| DROP FOREIGN KEY fk_symbol
| DISABLE KEYS
| ENABLE KEYS
| RENAME [TO] new_tbl_name
| ORDER BY col_name
| CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
| [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
| DISCARD TABLESPACE
| IMPORT TABLESPACE
| table_option ...

```

index_col_name:
col_name [(*length*)] [ASC | DESC]

index_type:
 USING {BTREE | HASH}

ALTER TABLE enables you to change the structure of an existing table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change the comment for the table and type of the table.

The syntax for many of the allowable alterations is similar to clauses of the CREATE TABLE statement. This includes *table_option* modifications, for options such as ENGINE, AUTO_INCREMENT, and AVG_ROW_LENGTH. (However, ALTER TABLE ignores the DATA DIRECTORY and INDEX DIRECTORY table options.)

[Section 13.1.5, “CREATE TABLE Syntax”](#), lists all table options. As of MySQL 5.0.23, to prevent inadvertent loss of data, ALTER TABLE cannot be used to change the storage engine of a table to MERGE or BLACKHOLE.

Some operations may result in warnings if attempted on a table for which the storage engine does not support the operation. These warnings can be displayed with SHOW WARNINGS. See [Section 13.5.4.25, “SHOW WARNINGS Syntax”](#).

If you use ALTER TABLE to change a column specification but DESCRIBE *tbl_name* indicates that your column was not changed, it is possible that MySQL ignored your modification for one of the reasons described in [Section 13.1.5.1, “Silent Column Specification Changes”](#).

In most cases, ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, and then the original table is deleted and the new one is renamed. While ALTER TABLE is executing, the original table is readable by other clients. Updates and writes to the table are stalled until the new table is ready, and then are automatically redirected to the new table without any failed updates.

If you use ALTER TABLE *tbl_name* RENAME TO *new_tbl_name* without any other options, MySQL simply renames any files that correspond to the table *tbl_name*. There is no need to create a temporary table. (You can also use the RENAME TABLE statement to rename tables. See [Section 13.1.9, “RENAME TABLE Syntax”](#).)

If you use any option to ALTER TABLE other than RENAME, MySQL always creates a temporary table, even if the data wouldn't strictly need to be copied (such as when you change the name of a column). For MyISAM tables, you can speed up the index re-creation operation (which is the slowest part of the alteration process) by setting the `myisam_sort_buffer_size` system variable to a high value.

- To use ALTER TABLE, you need ALTER, INSERT, and CREATE privileges for the table.
- IGNORE is a MySQL extension to standard SQL. It controls how ALTER TABLE works if there are duplicates on unique keys in the new table or if warnings occur when strict mode is enabled. If IGNORE is not specified, the copy is aborted and rolled back if duplicate-key errors occur. If IGNORE is specified, only the first row is used of rows with duplicates on a unique key. The other conflicting rows are deleted. Incorrect values are truncated to the closest matching acceptable value.
- You can issue multiple ADD, ALTER, DROP, and CHANGE clauses in a single ALTER TABLE statement, separated by commas. This is a MySQL extension to standard SQL, which allows only one of each clause per ALTER TABLE statement. For example, to drop multiple columns in a single statement, do this:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- CHANGE *col_name*, DROP *col_name*, and DROP INDEX are MySQL extensions

to standard SQL.

- `MODIFY` is an Oracle extension to `ALTER TABLE`.
- The word `COLUMN` is optional and can be omitted.
- *column_definition* clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. Note that this syntax includes the column name, not just its data type. See [Section 13.1.5, “CREATE TABLE Syntax”](#).

- You can rename a column using a `CHANGE old_col_name column_definition` clause. To do so, specify the old and new column names and the type that the column currently has. For example, to rename an `INTEGER` column from `a` to `b`, you can do this:

```
ALTER TABLE t1 CHANGE a b INTEGER;
```

If you want to change a column's type but not the name, `CHANGE` syntax still requires an old and new column name, even if they are the same. For example:

```
ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

You can also use `MODIFY` to change a column's type without renaming it:

```
ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.
- When you change a data type using `CHANGE` or `MODIFY`, MySQL tries to convert existing column values to the new type as well as possible.
- To add a column at a specific position within a table row, use `FIRST` or `AFTER col_name`. The default is to add the column last. You can also use `FIRST` and `AFTER` in `CHANGE` or `MODIFY` operations.
- `ALTER ... SET DEFAULT` or `ALTER ... DROP DEFAULT` specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be `NULL`, the new default is

NULL. If the column cannot be NULL, MySQL assigns a default value, as described in [Section 11.1.4, “Data Type Default Values”](#).

- `DROP INDEX` removes an index. This is a MySQL extension to standard SQL. See [Section 13.1.7, “DROP INDEX Syntax”](#).
- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well.
- If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use `DROP TABLE` instead.
- `DROP PRIMARY KEY` drops the primary index. *Note:* In older versions of MySQL, if no primary index existed, `DROP PRIMARY KEY` would drop the first `UNIQUE` index in the table. This is not the case in MySQL 5.0, where trying to use `DROP PRIMARY KEY` on a table with no primary key results in an error.

If you add a `UNIQUE INDEX` or `PRIMARY KEY` to a table, it is stored before any non-unique index so that MySQL can detect duplicate keys as early as possible.

- Some storage engines allow you to specify an index type when creating an index. The syntax for the *index_type* specifier is `USING type_name`. For details about `USING`, see [Section 13.1.4, “CREATE INDEX Syntax”](#).
- `ORDER BY` enables you to create the new table with the rows in a specific order. Note that the table does not remain in this order after inserts and deletes. This option is useful primarily when you know that you are mostly to query the rows in a certain order most of the time. By using this option after major changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.
- If you use `ALTER TABLE` on a MyISAM table, all non-unique indexes are created in a separate batch (as for `REPAIR TABLE`). This should make `ALTER TABLE` much faster when you have many indexes.

This feature can be activated explicitly. `ALTER TABLE ... DISABLE KEYS`

tells MySQL to stop updating non-unique indexes for a MyISAM table. `ALTER TABLE ... ENABLE KEYS` then should be used to re-create missing indexes. MySQL does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using `ALTER TABLE ... DISABLE KEYS` requires the `INDEX` privilege in addition to the privileges mentioned earlier.

- The `FOREIGN KEY` and `REFERENCES` clauses are supported by the InnoDB storage engine, which implements `ADD [CONSTRAINT [symbol]] FOREIGN KEY (...) REFERENCES ... (...)`. See [Section 14.2.6.4, “FOREIGN KEY Constraints”](#). For other storage engines, the clauses are parsed but ignored. The `CHECK` clause is parsed but ignored by all storage engines. See [Section 13.1.5, “CREATE TABLE Syntax”](#). The reason for accepting but ignoring syntax clauses is for compatibility, to make it easier to port code from other SQL servers, and to run applications that create tables with references. See [Section 1.9.5, “MySQL Differences from Standard SQL”](#).

You cannot add a foreign key and drop a foreign key in separate clauses of a single `ALTER TABLE` statement. You must use separate statements.

- InnoDB supports the use of `ALTER TABLE` to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

You cannot add a foreign key and drop a foreign key in separate clauses of a single `ALTER TABLE` statement. You must use separate statements.

For more information, see [Section 14.2.6.4, “FOREIGN KEY Constraints”](#).

- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.
- If you want to change the table default character set and all character columns (`CHAR`, `VARCHAR`, `TEXT`) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

Warning: The preceding operation converts column values between the

character sets. This is *not* what you want if you have a column in one character set (like `latin1`) but the stored values actually use some other, incompatible character set (like `utf8`). In this case, you have to do the following for each such column:

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;  
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

The reason this works is that there is no conversion when you convert to or from BLOB columns.

If you specify `CONVERT TO CHARACTER SET binary`, the `CHAR`, `VARCHAR`, and `TEXT` columns are converted to their corresponding binary string types (`BINARY`, `VARBINARY`, `BLOB`). This means that the columns no longer will have a character set and a subsequent `CONVERT TO` operation will not apply to them.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word `DEFAULT` is optional. The default character set is the character set that is used if you do not specify the character set for a new column which you add to a table (for example, with `ALTER TABLE ... ADD column`).

- For an InnoDB table that is created with its own tablespace in an `.ibd` file, that file can be discarded and imported. To discard the `.ibd` file, use this statement:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

This deletes the current `.ibd` file, so be sure that you have a backup first. Attempting to access the table while the tablespace file is discarded results in an error.

To import the backup `.ibd` file back into the table, copy it into the database directory, and then issue this statement:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

See [Section 14.2.3.1, “Using Per-Table Tablespaces”](#).

With the `mysql_info()` C API function, you can find out how many rows were copied, and (when `IGNORE` is used) how many rows were deleted due to duplication of unique key values. See [Section 22.2.3.34, “mysql_info\(\)”](#).

Here are some examples that show uses of `ALTER TABLE`. Begin with a table `t1` that is created as shown here:

```
CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from `t1` to `t2`:

```
ALTER TABLE t1 RENAME t2;
```

To change column `a` from `INTEGER` to `TINYINT NOT NULL` (leaving the name the same), and to change column `b` from `CHAR(10)` to `CHAR(20)` as well as renaming it from `b` to `c`:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new `TIMESTAMP` column named `d`:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

To add indexes on column `d` and on column `a`:

```
ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
```

To remove column `c`:

```
ALTER TABLE t2 DROP COLUMN c;
```

To add a new `AUTO_INCREMENT` integer column named `c`:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,  
ADD PRIMARY KEY (c);
```

Note that we indexed `c` (as a `PRIMARY KEY`), because `AUTO_INCREMENT` columns must be indexed, and also that we declare `c` as `NOT NULL`, because primary key columns cannot be `NULL`.

When you add an `AUTO_INCREMENT` column, column values are filled in with sequence numbers for you automatically. For `MyISAM` tables, you can set the first

sequence number by executing `SET INSERT_ID=value` before `ALTER TABLE` or by using the `AUTO_INCREMENT=value` table option. See [Section 13.5.3, “SET Syntax”](#).

From MySQL 5.0.3, you can use the `ALTER TABLE ... AUTO_INCREMENT=value` table option for InnoDB tables to set the sequence number for new rows if the value is greater than the maximum value in the `AUTO_INCREMENT` column. *If the value is less than the current maximum value in the column, no error message is given and the current sequence value is not changed.*

With MyISAM tables, if you do not change the `AUTO_INCREMENT` column, the sequence number is not affected. If you drop an `AUTO_INCREMENT` column and then add another `AUTO_INCREMENT` column, the numbers are resequenced beginning with 1.

When replication is used, adding an `AUTO_INCREMENT` column to a table might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to the table `t1`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table `t1` has columns `col1` and `col2`.

This set of statements will also produce a new table `t2` identical to `t1`, with the addition of an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE T2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important: To guarantee the same ordering on both master and slave, *all* columns of `t1` must be referenced in the `ORDER BY` clause.

Regardless of the method used to create and populate the copy having the

AUTO_INCREMENT column, the final step is to drop the original table and then rename the copy:

```
DROP t1;  
ALTER TABLE t2 RENAME t1;
```

See also [Section A.7.1, “Problems with ALTER TABLE”](#).

13.1.3. CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name  
    [create_specification [, create_specification] ...]
```

```
create_specification:  
    [DEFAULT] CHARACTER SET charset_name  
    | [DEFAULT] COLLATE collation_name
```

CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE privilege for the database. CREATE SCHEMA is a synonym for CREATE DATABASE as of MySQL 5.0.2.

An error occurs if the database exists and you did not specify IF NOT EXISTS.

create_specification options specify database characteristics. Database characteristics are stored in the `db.opt` file in the database directory. The CHARACTER SET clause specifies the default database character set. The COLLATE clause specifies the default database collation. [Chapter 10, Character Set Support](#), discusses character set and collation names.

A database in MySQL is implemented as a directory containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the CREATE DATABASE statement creates only a directory under the MySQL data directory and the `db.opt` file. Rules for allowable database names are given in [Section 9.2, “Database, Table, Index, Column, and Alias Names”](#).

If you manually create a directory under the data directory (for example, with `mkdir`), the server considers it a database directory and it shows up in the output of SHOW DATABASES.

You can also use the `mysqladmin` program to create databases. See [Section 8.9](#),

[“mysqladmin — Client for Administering a MySQL Server”](#).

13.1.4. CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
      [index_type]
      ON tbl_name (index_col_name, ...)
```

```
index_col_name:
  col_name [(length)] [ASC | DESC]
```

```
index_type:
  USING {BTREE | HASH}
```

CREATE INDEX is mapped to an ALTER TABLE statement to create indexes. See [Section 13.1.2, “ALTER TABLE Syntax”](#). For more information about indexes, see [Section 7.4.5, “How MySQL Uses Indexes”](#).

Normally, you create all indexes on a table at the time the table itself is created with CREATE TABLE. See [Section 13.1.5, “CREATE TABLE Syntax”](#). CREATE INDEX enables you to add indexes to existing tables.

A column list of the form (*col1*, *col2*, ...) creates a multiple-column index. Index values are formed by concatenating the values of the given columns.

For CHAR, VARCHAR, BINARY, and VARBINARY columns, indexes can be created that use only the leading part of column values, using *col_name(length)* syntax to specify an index prefix length. BLOB and TEXT columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for non-binary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for CHAR, VARCHAR, and TEXT columns, and the first *length* bytes of each column value for BINARY, VARBINARY, and BLOB columns.

The statement shown here creates an index using the first 10 characters of the name column:

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, this index should not be much slower than an index created from the entire name column. Also, using partial columns for indexes can make the index file much smaller, which

could save a lot of disk space and might also speed up INSERT operations.

Prefixes can be up to 1000 bytes long (767 bytes for InnoDB tables). Note that prefix limits are measured in bytes, whereas the prefix length in CREATE INDEX statements is interpreted as number of characters for non-binary data types (CHAR, VARCHAR, TEXT). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. This constraint does not apply to NULL values except for the BDB storage engine. For other engines, a UNIQUE index allows multiple NULL values for columns that can contain NULL.

FULLTEXT indexes are supported only for MyISAM tables and can include only CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; partial indexing is not supported and any prefix length is ignored if specified. See [Section 12.7, “Full-Text Search Functions”](#), for details of operation.

SPATIAL indexes are supported only for MyISAM tables and can include only spatial columns that are defined as NOT NULL. [Chapter 16, Spatial Extensions](#), describes the spatial data types.

In MySQL 5.0:

- You can add an index on a column that can have NULL values only if you are using the MyISAM, InnoDB, BDB, or MEMORY storage engine.
- You can add an index on a BLOB or TEXT column only if you are using the MyISAM, BDB, or InnoDB storage engine.

An *index_col_name* specification can end with ASC or DESC. These keywords are allowed for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

Some storage engines allow you to specify an index type when creating an index. The allowable index type values supported by different storage engines are shown in the following table. Where multiple index types are listed, the first

one is the default when no index type specifier is given.

Storage Engine	Allowable Index Types
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE

If you specify an index type that is not legal for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.

Examples:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;  
CREATE INDEX id_index USING BTREE ON lookup (id);
```

TYPE *type_name* is recognized as a synonym for USING *type_name*. However, USING is the preferred form.

13.1.5. CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
    (create_definition,...)  
    [table_option ...]
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
    [(create_definition,...)]  
    [table_option ...]  
    select_statement
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name  
    { LIKE old_tbl_name | (LIKE old_tbl_name) }
```

create_definition:

column_definition

```
| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,.  
| {INDEX|KEY} [index_name] [index_type] (index_col_name,...)  
| [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]  
    [index_name] [index_type] (index_col_name,...)
```

```
| {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)  
| [CONSTRAINT [symbol]] FOREIGN KEY  
|   [index_name] (index_col_name,...) [reference_definition]  
| CHECK (expr)
```

column_definition:

```
  col_name data_type [NOT NULL | NULL] [DEFAULT default_value]  
    [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]  
    [COMMENT 'string'] [reference_definition]
```

data_type:

```
  BIT[(length)]  
| TINYINT[(length)] [UNSIGNED] [ZEROFILL]  
| SMALLINT[(length)] [UNSIGNED] [ZEROFILL]  
| MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]  
| INT[(length)] [UNSIGNED] [ZEROFILL]  
| INTEGER[(length)] [UNSIGNED] [ZEROFILL]  
| BIGINT[(length)] [UNSIGNED] [ZEROFILL]  
| REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]  
| DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]  
| FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]  
| DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]  
| NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]  
| DATE  
| TIME  
| TIMESTAMP  
| DATETIME  
| YEAR  
| CHAR(length)  
|   [CHARACTER SET charset_name] [COLLATE collation_name]  
| VARCHAR(length)  
|   [CHARACTER SET charset_name] [COLLATE collation_name]  
| BINARY(length)  
| VARBINARY(length)  
| TINYBLOB  
| BLOB  
| MEDIUMBLOB  
| LONGBLOB  
| TINYTEXT [BINARY]  
|   [CHARACTER SET charset_name] [COLLATE collation_name]  
| TEXT [BINARY]  
|   [CHARACTER SET charset_name] [COLLATE collation_name]  
| MEDIUMTEXT [BINARY]  
|   [CHARACTER SET charset_name] [COLLATE collation_name]  
| LONGTEXT [BINARY]  
|   [CHARACTER SET charset_name] [COLLATE collation_name]  
| ENUM(value1,value2,value3,...) [COLLATE collation_name]  
| SET(value1,value2,value3,...) [COLLATE collation_name]
```

```

| spatial_type

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

reference_definition:
    REFERENCES tbl_name [(index_col_name,...)]
        [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
        [ON DELETE reference_option]
        [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION

table_option:
    {ENGINE|TYPE} [=] engine_name
| AUTO_INCREMENT [=] value
| AVG_ROW_LENGTH [=] value
| [DEFAULT] CHARACTER SET charset_name
| CHECKSUM [=] {0 | 1}
| COLLATE collation_name
| COMMENT [=] 'string'
| CONNECTION [=] 'connect_string'
| DATA DIRECTORY [=] 'absolute path to directory'
| DELAY_KEY_WRITE [=] {0 | 1}
| INDEX DIRECTORY [=] 'absolute path to directory'
| INSERT_METHOD [=] { NO | FIRST | LAST }
| MAX_ROWS [=] value
| MIN_ROWS [=] value
| PACK_KEYS [=] {0 | 1 | DEFAULT}
| PASSWORD [=] 'string'
| ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPA
| UNION [=] (tbl_name[,tbl_name]...)

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ... (Some legal select statemen

```

CREATE TABLE creates a table with the given name. You must have the CREATE privilege for the table.

Rules for allowable table names are given in [Section 9.2, “Database, Table, Index, Column, and Alias Names”](#). By default, the table is created in the default database. An error occurs if the table exists, if there is no default database, or if the database does not exist.

The table name can be specified as *db_name.tbl_name* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists. If you use quoted identifiers, quote the database and table names separately. For example, write ``mydb`.`mytbl``, not ``mydb.mytbl``.

You can use the `TEMPORARY` keyword when creating a table. A `TEMPORARY` table is visible only to the current connection, and is dropped automatically when the connection is closed. This means that two different connections can use the same temporary table name without conflicting with each other or with an existing non-`TEMPORARY` table of the same name. (The existing table is hidden until the temporary table is dropped.) To create temporary tables, you must have the `CREATE TEMPORARY TABLES` privilege.

The keywords `IF NOT EXISTS` prevent an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the `CREATE TABLE` statement. *Note:* If you use `IF NOT EXISTS` in a `CREATE TABLE . . . SELECT` statement, any rows selected by the `SELECT` part are inserted regardless of whether the table already exists.

MySQL represents each table by an `.frm` table format (definition) file in the database directory. The storage engine for the table might create other files as well. In the case of `MyISAM` tables, the storage engine creates data and index files. Thus, for each `MyISAM` table *tbl_name*, there are three disk files:

File	Purpose
<code>tbl_name.frm</code>	Table format (definition) file
<code>tbl_name.MYD</code>	Data file
<code>tbl_name.MYI</code>	Index file

[Chapter 14, Storage Engines and Table Types](#), describes what files each storage engine creates to represent tables.

data_type represents the data type is a column definition. *spatial_type* represents a spatial data type. For general information on the properties of data types other than the spatial types, see [Chapter 11, Data Types](#). For information about spatial data types, see [Chapter 16, Spatial Extensions](#).

Some attributes do not apply to all data types. `AUTO_INCREMENT` applies only to integer types. `DEFAULT` does not apply to the `BLOB` or `TEXT` types.

- If neither `NULL` nor `NOT NULL` is specified, the column is treated as though `NULL` had been specified.
- An integer column can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is `value+1`, where *value* is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with 1.

To retrieve an `AUTO_INCREMENT` value after inserting a row, use the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. See [Section 12.9.3, “Information Functions”](#), and [Section 22.2.3.36, “mysql_insert_id\(\)”](#).

If the `NO_AUTO_VALUE_ON_ZERO` SQL mode is enabled, you can store `0` in `AUTO_INCREMENT` columns as `0` without generating a new sequence value. See [Section 5.2.5, “The Server SQL Mode”](#).

Note: There can be only one `AUTO_INCREMENT` column per table, it must be indexed, and it cannot have a `DEFAULT` value. An `AUTO_INCREMENT` column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers “wrap” over from positive to negative and also to ensure that you do not accidentally get an `AUTO_INCREMENT` column that contains `0`.

For `MyISAM` and `BDB` tables, you can specify an `AUTO_INCREMENT` secondary column in a multiple-column key. See [Section 3.6.9, “Using AUTO_INCREMENT”](#).

To make MySQL compatible with some ODBC applications, you can find the `AUTO_INCREMENT` value for the last inserted row with the following query:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

For information about `InnoDB` and `AUTO_INCREMENT`, see [Section 14.2.6.3](#),

[“How AUTO INCREMENT Columns Work in InnoDB”](#).

- The attribute SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.
- Character data types (CHAR, VARCHAR, TEXT) can include CHARACTER SET and COLLATE attributes to specify the character set and collation for the column. For details, see [Chapter 10, Character Set Support](#). CHARSET is a synonym for CHARACTER SET. Example:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

MySQL 5.0 interprets length specifications in character column definitions in characters. (Versions before MySQL 4.1 interpreted them in bytes.) Lengths for BINARY and VARBINARY are in bytes.

- The DEFAULT clause specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as NOW() or CURRENT_DATE. The exception is that you can specify CURRENT_TIMESTAMP as the default for a TIMESTAMP column. See [Section 11.3.1.1, “TIMESTAMP Properties as of MySQL 4.1”](#).

If a column definition includes no explicit DEFAULT value, MySQL determines the default value as described in [Section 11.1.4, “Data Type Default Values”](#).

BLOB and TEXT columns cannot be assigned a default value.

- A comment for a column can be specified with the COMMENT option, up to 255 characters long. The comment is displayed by the SHOW CREATE TABLE and SHOW FULL COLUMNS statements.
- KEY is normally a synonym for INDEX. The key attribute PRIMARY KEY can also be specified as just KEY when given in a column definition. This was implemented for compatibility with other database systems.
- A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that

matches an existing row. This constraint does not apply to NULL values except for the BDB storage engine. For other engines, a UNIQUE index allows multiple NULL values for columns that can contain NULL.

- A PRIMARY KEY is a unique index where all key columns must be defined as NOT NULL. If they are not explicitly declared as NOT NULL, MySQL declares them so implicitly (and silently). A table can have only one PRIMARY KEY. If you do not have a PRIMARY KEY and an application asks for the PRIMARY KEY in your tables, MySQL returns the first UNIQUE index that has no NULL columns as the PRIMARY KEY.

In InnoDB tables, having a long PRIMARY KEY wastes a lot of space. (See [Section 14.2.13, “InnoDB Table and Index Structures”](#).)

- In the created table, a PRIMARY KEY is placed first, followed by all UNIQUE indexes, and then the non-unique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated UNIQUE keys.
- A PRIMARY KEY can be a multiple-column index. However, you cannot create a multiple-column index using the PRIMARY KEY key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate PRIMARY KEY(index_col_name, ...) clause.
- If a PRIMARY KEY or UNIQUE index consists of only one column that has an integer type, you can also refer to the column as `_rowid` in SELECT statements.
- In MySQL, the name of a PRIMARY KEY is PRIMARY. For other indexes, if you do not assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (`_2`, `_3`, ...) to make it unique. You can see index names for a table using `SHOW INDEX FROM tbl_name`. See [Section 13.5.4.13, “SHOW INDEX Syntax”](#).
- Some storage engines allow you to specify an index type when creating an index. The syntax for the *index_type* specifier is `USING type_name`.

Example:

```
CREATE TABLE lookup
```

```
(id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

For details about USING, see [Section 13.1.4, “CREATE INDEX Syntax”](#).

For more information about indexes, see [Section 7.4.5, “How MySQL Uses Indexes”](#).

- In MySQL 5.0, only the MyISAM, InnoDB, BDB, and MEMORY storage engines support indexes on columns that can have NULL values. In other cases, you must declare indexed columns as NOT NULL or an error results.
- For CHAR, VARCHAR, BINARY, and VARBINARY columns, indexes can be created that use only the leading part of column values, using `col_name(length)` syntax to specify an index prefix length. BLOB and TEXT columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for non-binary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for CHAR, VARCHAR, and TEXT columns, and the first *length* bytes of each column value for BINARY, VARBINARY, and BLOB columns. Indexing only a prefix of column values like this can make the index file much smaller. See [Section 7.4.3, “Column Indexes”](#).

Only the MyISAM, BDB, and InnoDB storage engines support indexing on BLOB and TEXT columns. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for InnoDB tables). Note that prefix limits are measured in bytes, whereas the prefix length in CREATE TABLE statements is interpreted as number of characters for non-binary data types (CHAR, VARCHAR, TEXT). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

- An *index_col_name* specification can end with ASC or DESC. These keywords are allowed for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.
- When you use ORDER BY or GROUP BY on a TEXT or BLOB column in a

SELECT, the server sorts values using only the initial number of bytes indicated by the `max_sort_length` system variable. See [Section 11.4.3, “The BLOB and TEXT Types”](#).

- You can create special FULLTEXT indexes, which are used for full-text searches. Only the MyISAM storage engine supports FULLTEXT indexes. They can be created only from CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; partial indexing is not supported and any prefix length is ignored if specified. See [Section 12.7, “Full-Text Search Functions”](#), for details of operation.
- You can create SPATIAL indexes on spatial data types. Spatial types are supported only for MyISAM tables and indexed columns must be declared as NOT NULL. See [Chapter 16, Spatial Extensions](#).
- InnoDB tables support checking of foreign key constraints. See [Section 14.2, “The InnoDB Storage Engine”](#). Note that the FOREIGN KEY syntax in InnoDB is more restrictive than the syntax presented for the CREATE TABLE statement at the beginning of this section: The columns of the referenced table must always be explicitly named. InnoDB supports both ON DELETE and ON UPDATE actions on foreign keys. For the precise syntax, see [Section 14.2.6.4, “FOREIGN KEY Constraints”](#).

For other storage engines, MySQL Server parses and ignores the FOREIGN KEY and REFERENCES syntax in CREATE TABLE statements. The CHECK clause is parsed but ignored by all storage engines. See [Section 1.9.5.5, “Foreign Keys”](#).

- For MyISAM tables, each NULL column takes one bit extra, rounded up to the nearest byte. The maximum row length in bytes can be calculated as follows:

```
row length = 1
              + (sum of column lengths)
              + (number of NULL columns + delete_flag + 7)/8
              + (number of variable-length columns)
```

delete_flag is 1 for tables with static row format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. *delete_flag* is 0 for dynamic tables because the flag is stored in the

dynamic row header.

These calculations do not apply for InnoDB tables, for which storage size is no different for NULL columns than for NOT NULL columns.

The ENGINE table option specifies the storage engine for the table. TYPE is a synonym, but ENGINE is the preferred option name.

The ENGINE table option takes the storage engine names shown in the following table.

Storage Engine	Description
ARCHIVE	The archiving storage engine. See Section 14.8, “The ARCHIVE Storage Engine” .
BDB	Transaction-safe tables with page locking. Also known as BerkeleyDB. See Section 14.5, “The BDB (BerkeleyDB) Storage Engine” .
CSV	Tables that store rows in comma-separated values format. See Section 14.9, “The csv Storage Engine” .
EXAMPLE	An example engine. See Section 14.6, “The EXAMPLE Storage Engine” .
FEDERATED	Storage engine that accesses remote tables. See Section 14.7, “The FEDERATED Storage Engine” .
HEAP	This is a synonym for MEMORY.
ISAM (OBSOLETE)	Not available in MySQL 5.0. If you are upgrading to MySQL 5.0 from a previous version, you should convert any existing ISAM tables to MyISAM <i>before</i> performing the upgrade.
InnoDB	Transaction-safe tables with row locking and foreign keys. See Section 14.2, “The InnoDB Storage Engine” .
MEMORY	The data for this storage engine is stored only in memory. See Section 14.4, “The MEMORY (HEAP) Storage Engine” .
MERGE	A collection of MyISAM tables used as one table. Also known as MRG_MyISAM. See Section 14.3, “The MERGE Storage Engine” .
MyISAM	The binary portable storage engine that is the default storage engine used by MySQL. See Section 14.1, “The MyISAM Storage

	Engine ".
NDBCLUSTER	Clustered, fault-tolerant, memory-based tables. Also known as NDB. See Chapter 15, MySQL Cluster .

If a storage engine is specified that is not available, MySQL uses the default engine instead. Normally, this is MyISAM. For example, if a table definition includes the ENGINE=BDB option but the MySQL server does not support BDB tables, the table is created as a MyISAM table. This makes it possible to have a replication setup where you have transactional tables on the master but tables created on the slave are non-transactional (to get more speed). In MySQL 5.0, a warning occurs if the storage engine specification is not honored.

The other table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them. These options apply to all storage engines unless otherwise indicated. Options that do not apply to a given storage engine may be accepted and remembered as part of the table definition. Such options then apply if you later use ALTER TABLE to convert the table to use a different storage engine.

- AUTO_INCREMENT

The initial AUTO_INCREMENT value for the table. In MySQL 5.0, this works for MyISAM and MEMORY tables. It is also supported for InnoDB as of MySQL 5.0.3. To set the first auto-increment value for engines that do not support the AUTO_INCREMENT table option, insert a “dummy” row with a value one less than the desired value after creating the table, and then delete the dummy row.

For engines that support the AUTO_INCREMENT table option in CREATE TABLE statements, you can also use ALTER TABLE tbl_name AUTO_INCREMENT = N to reset the AUTO_INCREMENT value.

- AVG_ROW_LENGTH

An approximation of the average row length for your table. You need to set this only for large tables with variable-size rows.

When you create a MyISAM table, MySQL uses the product of the MAX_ROWS and AVG_ROW_LENGTH options to decide how big the resulting table is. If you

don't specify either option, the maximum size for a table is 65,536TB of data (4GB before MySQL 5.0.6). (If your operating system does not support files that large, table sizes are constrained by the file size limit.) If you want to keep down the pointer sizes to make the index smaller and faster and you don't really need big files, you can decrease the default pointer size by setting the `myisam_data_pointer_size` system variable, which was added in MySQL 4.1.2. (See [Section 5.2.2, “Server System Variables”](#).) If you want all your tables to be able to grow above the default limit and are willing to have your tables slightly slower and larger than necessary, you can increase the default pointer size by setting this variable.

- [DEFAULT] CHARACTER SET

Specify a default character set for the table. `CHARSET` is a synonym for `CHARACTER SET`.

- CHECKSUM

Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update, but also makes it easier to find corrupted tables. The `CHECKSUM TABLE` statement reports the checksum. (MyISAM only.)

- COLLATE

Specify a default collation for the table.

- COMMENT

A comment for the table, up to 60 characters long.

- CONNECTION

The connection string for a `FEDERATED` table. This option is available as of MySQL 5.0.13; before that, use a `COMMENT` option for the connection string.

- DATA DIRECTORY, INDEX DIRECTORY

By using `DATA DIRECTORY='directory'` or `INDEX DIRECTORY='directory'`

you can specify where the MyISAM storage engine should put a table's data file and index file. The directory must be the full pathname to the directory, not a relative path.

These options work only when you are not using the `--skip-symbolic-links` option. Your operating system must also have a working, thread-safe `realpath()` call. See [Section 7.6.1.2, “Using Symbolic Links for Tables on Unix”](#), for more complete information.

- `DELAY_KEY_WRITE`

Set this to 1 if you want to delay key updates for the table until the table is closed. See the description of the `delay_key_write` system variable in [Section 5.2.2, “Server System Variables”](#). (MyISAM only.)

- `INSERT_METHOD`

If you want to insert data into a MERGE table, you must specify with `INSERT_METHOD` the table into which the row should be inserted. `INSERT_METHOD` is an option useful for MERGE tables only. Use a value of `FIRST` or `LAST` to have inserts go to the first or last table, or a value of `NO` to prevent inserts. See [Section 14.3, “The MERGE Storage Engine”](#).

- `MAX_ROWS`

The maximum number of rows you plan to store in the table. This is not a hard limit, but rather a hint to the storage engine that the table must be able to store at least this many rows.

- `MIN_ROWS`

The minimum number of rows you plan to store in the table.

- `PACK_KEYS`

`PACK_KEYS` takes effect only with MyISAM tables. Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of keys. Setting it to `DEFAULT` tells the storage engine to pack only long `CHAR` or `VARCHAR` columns.

If you do not use `PACK_KEYS`, the default is to pack strings, but not numbers. If you use `PACK_KEYS=1`, numbers are packed as well.

When packing binary number keys, MySQL uses prefix compression:

- Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.
- The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

This means that if you have many equal keys on two consecutive rows, all following “same” keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes `storage_size_for_key + pointer_size` (where the pointer size is usually 4). Conversely, you get a significant benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key is not a key that can have `NULL` values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is `NULL`.)

- **PASSWORD**

Encrypt the `.frm` file with a password. This option does nothing in the standard MySQL version.

- **ROW_FORMAT**

Defines how the rows should be stored. For `MyISAM` tables, the option value can be `FIXED` or `DYNAMIC` for static or variable-length row format.

myisampack sets the type to `COMPRESSED`. See [Section 14.1.3, “MyISAM Table Storage Formats”](#).

Starting with MySQL 5.0.3, for `InnoDB` tables, rows are stored in compact format (`ROW_FORMAT=COMPACT`) by default. The non-compact format used in older versions of MySQL can still be requested by specifying `ROW_FORMAT=REDUNDANT`.

- **RAID_TYPE**

RAID support has been removed as of MySQL 5.0. For information on RAID, see <http://dev.mysql.com/doc/refman/4.1/en/create-table.html>.

- UNION

UNION is used when you want to access a collection of identical MyISAM tables as one. This works only with MERGE tables. See [Section 14.3, “The MERGE Storage Engine”](#).

You must have SELECT, UPDATE, and DELETE privileges for the tables you map to a MERGE table. (*Note:* Formerly, all tables used had to be in the same database as the MERGE table itself. This restriction no longer applies.)

You can create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the SELECT. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,  
-> PRIMARY KEY (a), KEY(b))  
-> ENGINE=MyISAM SELECT b,c FROM test2;
```

This creates a MyISAM table with three columns, a, b, and c. Notice that the columns from the SELECT statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;  
+----+  
| n |  
+----+  
| 1 |  
+----+
```

```
mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;  
Query OK, 1 row affected (0.02 sec)  
Records: 1 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM bar;  
+-----+----+  
| m      | n |  
+-----+----+  
| NULL   | 1 |
```

```
+-----+----+
1 row in set (0.00 sec)
```

For each row in table `foo`, a row is inserted in `bar` with the values from `foo` and default values for the new columns.

In a table resulting from `CREATE TABLE ... SELECT`, columns named only in the `CREATE TABLE` part come first. Columns named in both parts or only in the `SELECT` part come after that. The data type of `SELECT` columns can be overridden by also specifying the column in the `CREATE TABLE` part.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

`CREATE TABLE ... SELECT` does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the `SELECT` statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Some conversion of data types might occur. For example, the `AUTO_INCREMENT` attribute is not preserved, and `VARCHAR` columns can become `CHAR` columns.

When creating a table with `CREATE ... SELECT`, make sure to alias any function calls or expressions in the query. If you do not, the `CREATE` statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
  SELECT artist.name, COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work ON artist.id = work.artist_id
  GROUP BY artist.id;
```

You can also explicitly specify the data type for a generated column:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

Use `LIKE` to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

`CREATE TABLE ... LIKE` does not preserve any `DATA DIRECTORY` or `INDEX DIRECTORY` table options that were specified for the original table, or any foreign key definitions.

You can precede the `SELECT` by `IGNORE` or `REPLACE` to indicate how to handle rows that duplicate unique key values. With `IGNORE`, new rows that duplicate an existing row on a unique key value are discarded. With `REPLACE`, new rows replace rows that have the same unique key value. If neither `IGNORE` nor `REPLACE` is specified, duplicate unique key values result in an error.

To ensure that the binary log can be used to re-create the original tables, MySQL does not allow concurrent inserts during `CREATE TABLE ... SELECT`.

13.1.5.1. Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a `CREATE TABLE` or `ALTER TABLE` statement. These might be changes to a data type, to attributes associated with a data type, or to an index specification.

Possible data type changes are given in the following list. These occur prior to MySQL 5.0.3. As of 5.0.3, an error occurs if a column cannot be created using the specified data type.

- `VARCHAR` columns with a length less than four are changed to `CHAR`.
- If any column in a table has a variable length, the entire row becomes variable-length as a result. Therefore, if a table contains any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB`), all `CHAR` columns longer than three characters are changed to `VARCHAR` columns. This does not affect how you use the columns in any way; in MySQL, `VARCHAR` is just a different way to store characters. MySQL performs this conversion because it saves space and makes table operations faster. See [Chapter 14, Storage Engines and Table Types](#).
- Before MySQL 5.0.3, a `CHAR` or `VARCHAR` column with a length specification greater than 255 is converted to the smallest `TEXT` type that can hold values of the given length. For example, `VARCHAR(500)` is converted to `TEXT`, and `VARCHAR(200000)` is converted to `MEDIUMTEXT`. Note that this conversion results in a change in behavior with regard to treatment of trailing spaces.

Similar conversions occur for `BINARY` and `VARBINARY`, except that they are converted to a `BLOB` type.

Starting with MySQL 5.0.3, a `CHAR` or `BINARY` column with a length specification greater than 255 is not silently converted. Instead, an error occurs. From MySQL 5.0.6 on, silent conversion of `VARCHAR` and `VARBINARY` columns with a length specification greater than 65,535 does not occur if strict SQL mode is enabled. Instead, an error occurs.

- For a specification of `DECIMAL(M,D)`, if M is not larger than D , it is adjusted upward. For example, `DECIMAL(10,10)` becomes `DECIMAL(11,10)`.

Other silent column specification changes include changes to attribute or index specifications:

- `TIMESTAMP` display sizes are discarded. Note that `TIMESTAMP` columns have changed considerably in recent versions of MySQL prior to 5.0; for a description of these changes, see the MySQL 3.23, 4.0, 4.1 Reference Manual.
- Columns that are part of a `PRIMARY KEY` are made `NOT NULL` even if not declared that way.
- Trailing spaces are automatically deleted from `ENUM` and `SET` member values when the table is created.
- MySQL maps certain data types used by other SQL database vendors to MySQL types. See [Section 11.7, “Using Data Types from Other Database Engines”](#).
- If you include a `USING` clause to specify an index type that is not legal for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.

To see whether MySQL used a data type other than the one you specified, issue a `DESCRIBE` or `SHOW CREATE TABLE` statement after creating or altering the table.

Certain other data type changes can occur if you compress a table using `myisampack`. See [Section 14.1.3.3, “Compressed Table Characteristics”](#).

13.1.6. DROP DATABASE Syntax

`DROP {DATABASE | SCHEMA} [IF EXISTS] db_name`

`DROP DATABASE` drops all tables in the database and deletes the database. Be *very* careful with this statement! To use `DROP DATABASE`, you need the `DROP` privilege on the database. `DROP SCHEMA` is a synonym for `DROP DATABASE` as of MySQL 5.0.2.

`IF EXISTS` is used to prevent an error from occurring if the database does not exist.

If you use `DROP DATABASE` on a symbolically linked database, both the link and the original database are deleted.

`DROP DATABASE` returns the number of tables that were removed. This corresponds to the number of `.frm` files removed.

The `DROP DATABASE` statement removes from the given database directory those files and directories that MySQL itself may create during normal operation:

- All files with these extensions:

.BAK	.DAT	.HSH	.MRG
.MYD	.MYI	.TRG	.TRN
.db	.frm	.ibd	.ndb

- All subdirectories with names that consist of two hex digits `00-ff`. These are subdirectories used for RAID tables. (These directories are not removed as of MySQL 5.0, when support for RAID tables was removed. You should convert any existing RAID tables and remove these directories manually before upgrading to MySQL 5.0. See [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#).)
- The `db.opt` file, if it exists.

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the `DROP DATABASE` statement again.

You can also drop databases with **mysqladmin**. See [Section 8.9, “mysqladmin — Client for Administering a MySQL Server”](#).

13.1.7. DROP INDEX Syntax

```
DROP INDEX index_name ON tbl_name
```

DROP INDEX drops the index named *index_name* from the table *tbl_name*. This statement is mapped to an ALTER TABLE statement to drop the index. See [Section 13.1.2, “ALTER TABLE Syntax”](#).

13.1.8. DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]  
    tbl_name [, tbl_name] ...  
    [RESTRICT | CASCADE]
```

DROP TABLE removes one or more tables. You must have the DROP privilege for each table. All table data and the table definition are *removed*, so *be careful* with this statement! If any of the tables named in the argument list do not exist, MySQL returns an error indicating by name which non-existing tables it was unable to drop, but it also drops all of the tables in the list that do exist.

Use IF EXISTS to prevent an error from occurring for tables that do not exist. A NOTE is generated for each non-existent table when using IF EXISTS. See [Section 13.5.4.25, “SHOW WARNINGS Syntax”](#).

RESTRICT and CASCADE are allowed to make porting easier. For the moment, they do nothing.

Note: DROP TABLE automatically commits the current active transaction, unless you use the TEMPORARY keyword.

The TEMPORARY keyword has the following effects:

- The statement drops only TEMPORARY tables.
- The statement does not end an ongoing transaction.
- No access rights are checked. (A TEMPORARY table is visible only to the

client that created it, so no check is necessary.)

Using `TEMPORARY` is a good way to ensure that you do not accidentally drop a non-`TEMPORARY` table.

13.1.9. RENAME TABLE Syntax

```
RENAME TABLE tbl_name TO new_tbl_name  
    [, tbl_name2 TO new_tbl_name2] ...
```

This statement renames one or more tables.

The rename operation is done atomically, which means that no other thread can access any of the tables while the rename is running. For example, if you have an existing table `old_table`, you can create another table `new_table` that has the same structure but is empty, and then replace the existing table with the empty one as follows (assuming that `backup_table` does not already exist):

```
CREATE TABLE new_table (...);  
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

If the statement renames more than one table, renaming operations are done from left to right. If you want to swap two table names, you can do so like this (assuming that `tmp_table` does not already exist):

```
RENAME TABLE old_table TO tmp_table,  
             new_table TO old_table,  
             tmp_table TO new_table;
```

As long as two databases are on the same filesystem, you can use `RENAME TABLE` to move a table from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

As of MySQL 5.0.14, `RENAME TABLE` also works for views, as long as you do not try to rename a view into a different database.

When you execute `RENAME`, you cannot have any locked tables or active transactions. You must also have the `ALTER` and `DROP` privileges on the original table, and the `CREATE` and `INSERT` privileges on the new table.

If MySQL encounters any errors in a multiple-table rename, it does a reverse

rename for all renamed tables to return everything to its original state.

13.2. Data Manipulation Statements

13.2.1. DELETE Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
FROM table_references
    [WHERE where_condition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

For the single-table syntax, the DELETE statement deletes rows from *tbl_name* and returns the number of rows deleted. The WHERE clause, if given, specifies the conditions that identify which rows to delete. With no WHERE clause, all rows are deleted. If the ORDER BY clause is specified, the rows are deleted in the order that is specified. The LIMIT clause places a limit on the number of rows that can be deleted.

For the multiple-table syntax, DELETE deletes from each *tbl_name* the rows that satisfy the conditions. In this case, ORDER BY and LIMIT cannot be used.

where_condition is an expression that evaluates to true for each row to be deleted. It is specified as described in [Section 13.2.7, “SELECT Syntax”](#).

As stated, a DELETE statement with no WHERE clause deletes all rows. A faster way to do this, when you do not want to know the number of deleted rows, is to use TRUNCATE TABLE. See [Section 13.2.9, “TRUNCATE Syntax”](#).

If you delete the row containing the maximum value for an `AUTO_INCREMENT` column, the value is reused later for a BDB table, but not for a MyISAM or InnoDB table. If you delete all rows in the table with `DELETE FROM tbl_name` (without a `WHERE` clause) in `AUTO_COMMIT` mode, the sequence starts over for all storage engines except InnoDB and MyISAM. There are some exceptions to this behavior for InnoDB tables, as discussed in [Section 14.2.6.3, “How AUTO_INCREMENT Columns Work in InnoDB”](#).

For MyISAM and BDB tables, you can specify an `AUTO_INCREMENT` secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for MyISAM tables. See [Section 3.6.9, “Using AUTO_INCREMENT”](#).

The `DELETE` statement supports the following modifiers:

- If you specify `LOW_PRIORITY`, the server delays execution of the `DELETE` until no other clients are reading from the table.
- For MyISAM tables, if you use the `QUICK` keyword, the storage engine does not merge index leaves during delete, which may speed up some kinds of delete operations.
- The `IGNORE` keyword causes MySQL to ignore all errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of `OPTION` are returned as warnings.

The speed of delete operations may also be affected by factors discussed in [Section 7.2.18, “Speed of DELETE Statements”](#).

In MyISAM tables, deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions. To reclaim unused space and reduce file sizes, use the `OPTIMIZE TABLE` statement or the **`myisamchk`** utility to reorganize tables. `OPTIMIZE TABLE` is easier, but **`myisamchk`** is faster. See [Section 13.5.2.5, “OPTIMIZE TABLE Syntax”](#), and [Section 8.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

The `QUICK` modifier affects whether index leaves are merged for delete operations. `DELETE QUICK` is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In

this case, the holes left by deleted values are reused.

`DELETE QUICK` is not useful when deleted values lead to undef-filled index blocks spanning a range of index values for which new inserts occur again. In this case, use of `QUICK` can lead to wasted space in the index that remains unreclaimed.

Here is an example of such a scenario:

1. Create a table that contains an indexed `AUTO_INCREMENT` column.
2. Insert many rows into the table. Each insert results in an index value that is added to the high end of the index.
3. Delete a block of rows at the low end of the column range using `DELETE QUICK`.

In this scenario, the index blocks associated with the deleted index values become undef-filled but are not merged with other index blocks due to the use of `QUICK`. They remain undef-filled when new inserts occur, because new rows do not have index values in the deleted range. Furthermore, they remain undef-filled even if you later use `DELETE` without `QUICK`, unless some of the deleted index values happen to lie in index blocks within or adjacent to the undef-filled blocks. To reclaim unused index space under these circumstances, use `OPTIMIZE TABLE`.

If you are going to delete many rows from a table, it might be faster to use `DELETE QUICK` followed by `OPTIMIZE TABLE`. This rebuilds the index rather than performing many index block merge operations.

The MySQL-specific `LIMIT row_count` option to `DELETE` tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to ensure that a given `DELETE` statement does not take too much time. You can simply repeat the `DELETE` statement until the number of affected rows is less than the `LIMIT` value.

If the `DELETE` statement includes an `ORDER BY` clause, the rows are deleted in the order specified by the clause. This is really useful only in conjunction with `LIMIT`. For example, the following statement finds rows matching the `WHERE` clause, sorts them by `timestamp_column`, and deletes the first (oldest) one:

```
DELETE FROM some_log WHERE user = 'jcole'
```

```
ORDER BY timestamp_column LIMIT 1;
```

You can specify multiple tables in a `DELETE` statement to delete rows from one or more tables depending on the particular condition in the `WHERE` clause. However, you cannot use `ORDER BY` or `LIMIT` in a multiple-table `DELETE`. The *table_references* clause lists the tables involved in the join. Its syntax is described in [Section 13.2.7.1, “JOIN Syntax”](#).

For the first multiple-table syntax, only matching rows from the tables listed before the `FROM` clause are deleted. For the second multiple-table syntax, only matching rows from the tables listed in the `FROM` clause (before the `USING` clause) are deleted. The effect is that you can delete rows from many tables at the same time and have additional tables that are used only for searching:

```
DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.i
```

These statements use all three tables when searching for rows to delete, but delete matching rows only from tables `t1` and `t2`.

The preceding examples show inner joins that use the comma operator, but multiple-table `DELETE` statements can use any type of join allowed in `SELECT` statements, such as `LEFT JOIN`.

The syntax allows `.*` after the table names for compatibility with **Access**.

If you use a multiple-table `DELETE` statement involving InnoDB tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, you should delete from a single table and rely on the `ON DELETE` capabilities that InnoDB provides to cause the other tables to be modified accordingly.

Note: If you provide an alias for a table, you must use the alias when referring to the table:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

Cross-database deletes are supported for multiple-table deletes, but in this case, you must refer to the tables without using aliases. For example:

```
DELETE test1.tmp1, test2.tmp2 FROM test1.tmp1, test2.tmp2 WHERE ...
```

Currently, you cannot delete from a table and select from the same table in a subquery.

13.2.2. DO Syntax

```
DO expr [, expr] ...
```

DO executes the expressions but does not return any results. In most respects, DO is shorthand for SELECT *expr*, ..., but has the advantage that it is slightly faster when you do not care about the result.

DO is useful primarily with functions that have side effects, such as RELEASE_LOCK().

13.2.3. HANDLER Syntax

```
HANDLER tbl_name OPEN [ AS alias ]  
HANDLER tbl_name READ index_name { = | >= | <= | < } (value1,value2,  
    [ WHERE where_condition ] [LIMIT ... ]  
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }  
    [ WHERE where_condition ] [LIMIT ... ]  
HANDLER tbl_name READ { FIRST | NEXT }  
    [ WHERE where_condition ] [LIMIT ... ]  
HANDLER tbl_name CLOSE
```

The HANDLER statement provides direct access to table storage engine interfaces. It is available for MyISAM and InnoDB tables.

The HANDLER ... OPEN statement opens a table, making it accessible via subsequent HANDLER ... READ statements. This table object is not shared by other threads and is not closed until the thread calls HANDLER ... CLOSE or the thread terminates. If you open the table using an alias, further references to the open table with other HANDLER statements must use the alias rather than the table name.

The first HANDLER ... READ syntax fetches a row where the index specified

satisfies the given values and the `WHERE` condition is met. If you have a multiple-column index, specify the index column values as a comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index `my_idx` includes three columns named `col_a`, `col_b`, and `col_c`, in that order. The `HANDLER` statement can specify values for all three columns in the index, or for the columns in a leftmost prefix. For example:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

To employ the `HANDLER` interface to refer to a table's `PRIMARY KEY`, use the quoted identifier ``PRIMARY``:

```
HANDLER tbl_name READ `PRIMARY` ...
```

The second `HANDLER ... READ` syntax fetches a row from the table in index order that matches the `WHERE` condition.

The third `HANDLER ... READ` syntax fetches a row from the table in natural row order that matches the `WHERE` condition. It is faster than `HANDLER tbl_name READ index_name` when a full table scan is desired. Natural row order is the order in which rows are stored in a MyISAM table data file. This statement works for InnoDB tables as well, but there is no such concept because there is no separate data file.

Without a `LIMIT` clause, all forms of `HANDLER ... READ` fetch a single row if one is available. To return a specific number of rows, include a `LIMIT` clause. It has the same syntax as for the `SELECT` statement. See [Section 13.2.7, “SELECT Syntax”](#).

```
HANDLER ... CLOSE closes a table that was opened with HANDLER ... OPEN.
```

`HANDLER` is a somewhat low-level statement. For example, it does not provide consistency. That is, `HANDLER ... OPEN` does *not* take a snapshot of the table, and does *not* lock the table. This means that after a `HANDLER ... OPEN` statement is issued, table data can be modified (by the current thread or other threads) and these modifications might be only partially visible to `HANDLER ... NEXT` or `HANDLER ... PREV` scans.

There are several reasons to use the HANDLER interface instead of normal SELECT statements:

- HANDLER is faster than SELECT:
 - A designated storage engine handler object is allocated for the HANDLER . . . OPEN. The object is reused for subsequent HANDLER statements for that table; it need not be reinitialized for each one.
 - There is less parsing involved.
 - There is no optimizer or query-checking overhead.
 - The table does not have to be locked between two handler requests.
 - The handler interface does not have to provide a consistent look of the data (for example, dirty reads are allowed), so the storage engine can use optimizations that SELECT does not normally allow.
- For applications that use a low-level ISAM-like interface, HANDLER makes it much easier to port them to MySQL.
- HANDLER enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with SELECT. The HANDLER interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

13.2.4. INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name [(col_name,...)]
      VALUES ({expr | DEFAULT},...),(...),...
      [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name
      SET col_name={expr | DEFAULT}, ...
      [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
      [INTO] tbl_name [(col_name, ...)]
      SELECT ...
      [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

INSERT inserts new rows into an existing table. The INSERT ... VALUES and INSERT ... SET forms of the statement insert rows based on explicitly specified values. The INSERT ... SELECT form inserts rows selected from another table or tables. INSERT ... SELECT is discussed further in [Section 13.2.4.1, “INSERT ... SELECT Syntax”](#).

You can use REPLACE instead of INSERT to overwrite old rows. REPLACE is the counterpart to INSERT IGNORE in the treatment of new rows that contain unique key values that duplicate old rows: The new rows are used to replace the old rows rather than being discarded. See [Section 13.2.6, “REPLACE Syntax”](#).

tbl_name is the table into which rows should be inserted. The columns for which the statement provides values can be specified as follows:

- You can provide a comma-separated list of column names following the table name. In this case, a value for each named column must be provided by the VALUES list or the SELECT statement.
- If you do not specify a list of column names for INSERT ... VALUES or INSERT ... SELECT, values for every column in the table must be provided by the VALUES list or the SELECT statement. If you do not know the order of the columns in the table, use DESCRIBE *tbl_name* to find out.
- The SET clause indicates the column names explicitly.

Column values can be given in several ways:

- If you are not running in strict SQL mode, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in [Section 11.1.4, “Data Type Default Values”](#). See also [Section 1.9.6.2, “Constraints on Invalid Data”](#).

If you want an INSERT statement to generate an error unless you explicitly specify values for all columns that do not have a default value, you should

use strict mode. See [Section 5.2.5, “The Server SQL Mode”](#).

- Use the keyword `DEFAULT` to set a column explicitly to its default value. This makes it easier to write `INSERT` statements that assign values to all but a few columns, because it enables you to avoid writing an incomplete `VALUES` list that does not include a value for each column in the table. Otherwise, you would have to write out the list of column names corresponding to each value in the `VALUES` list.

You can also use `DEFAULT(col_name)` as a more general form that can be used in expressions to produce a given column's default value.

- If both the column list and the `VALUES` list are empty, `INSERT` creates a row with each column set to its default value:

```
INSERT INTO tbl_name () VALUES();
```

In strict mode, an error occurs if any column doesn't have a default value. Otherwise, MySQL uses the implicit default value for any column that does not have an explicitly defined default.

- You can specify an expression *expr* to provide a column value. This might involve type conversion if the type of the expression does not match the type of the column, and conversion of a given value can result in different inserted values depending on the data type. For example, inserting the string `'1999.0e-2'` into an `INT`, `FLOAT`, `DECIMAL(10,6)`, or `YEAR` column results in the values 1999, 19.9921, 19.992100, and 1999 being inserted, respectively. The reason the value stored in the `INT` and `YEAR` columns is 1999 is that the string-to-integer conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the floating-point and fixed-point columns, the string-to-floating-point conversion considers the entire string a valid floating-point value.

An expression *expr* can refer to any column that was set earlier in a value list. For example, you can do this because the value for `col2` refers to `col1`, which has previously been assigned:

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

But the following is not legal, because the value for `col1` refers to `col2`,

which is assigned after col1:

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

One exception involves columns that contain AUTO_INCREMENT values. Because the AUTO_INCREMENT value is generated after other value assignments, any reference to an AUTO_INCREMENT column in the assignment returns a 0.

INSERT statements that use VALUES syntax can insert multiple rows. To do this, include multiple lists of column values, each enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

The values list for each row must be enclosed within parentheses. The following statement is illegal because the number of values in the list does not match the number of column names:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

The rows-affected value for an INSERT can be obtained using the `mysql_affected_rows()` C API function. See [Section 22.2.3.1, “mysql_affected_rows\(\)”](#).

If you use an INSERT ... VALUES statement with multiple value lists or INSERT ... SELECT, the statement returns an information string in this format:

```
Records: 100 Duplicates: 0 Warnings: 0
```

Records indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because Duplicates can be non-zero.) Duplicates indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. Warnings indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting NULL into a column that has been declared NOT NULL. For multiple-row INSERT statements or INSERT INTO ... SELECT statements, the column is set to the implicit default value for the column data type. This is 0 for numeric types, the empty string (' ') for string types, and the “zero” value

for date and time types. `INSERT INTO ... SELECT` statements are handled the same way as multiple-row inserts because the server does not examine the result set from the `SELECT` to see whether it returns a single row. (For a single-row `INSERT`, no warning occurs when `NULL` is inserted into a `NOT NULL` column. Instead, the statement fails with an error.)

- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.
- Assigning a value such as '10.34 a' to a numeric column. The trailing non-numeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to 0.
- Inserting a string into a string column (`CHAR`, `VARCHAR`, `TEXT`, or `BLOB`) that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

If you are using the C API, the information string can be obtained by invoking the `mysql_info()` function. See [Section 22.2.3.34, “mysql_info\(\)”](#).

If `INSERT` inserts a row into a table that has an `AUTO_INCREMENT` column, you can find the value used for that column by using the SQL `LAST_INSERT_ID()` function. From within the C API, use the `mysql_insert_id()` function. However, you should note that the two functions do not always behave identically. The behavior of `INSERT` statements with respect to `AUTO_INCREMENT` columns is discussed further in [Section 12.9.3, “Information Functions”](#), and [Section 22.2.3.36, “mysql_insert_id\(\)”](#).

The `INSERT` statement supports the following modifiers:

- If you use the `DELAYED` keyword, the server puts the row or rows to be inserted into a buffer, and the client issuing the `INSERT DELAYED` statement can then continue immediately. If the table is in use, the server holds the rows. When the table is free, the server begins inserting rows, checking periodically to see whether there are any new read requests for the table. If there are, the delayed row queue is suspended until the table becomes free again. See [Section 13.2.4.2, “INSERT DELAYED Syntax”](#).

DELAYED is ignored with INSERT ... SELECT or INSERT ... ON DUPLICATE KEY UPDATE.

- If you use the LOW_PRIORITY keyword, execution of the INSERT is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the INSERT LOW_PRIORITY statement is waiting. It is possible, therefore, for a client that issues an INSERT LOW_PRIORITY statement to wait for a very long time (or even forever) in a read-heavy environment. (This is in contrast to INSERT DELAYED, which lets the client continue at once. Note that LOW_PRIORITY should normally not be used with MyISAM tables because doing so disables concurrent inserts. See [Section 7.3.3, “Concurrent Inserts”](#)).
- If you specify HIGH_PRIORITY, it overrides the effect of the --low-priority-updates option if the server was started with that option. It also causes concurrent inserts not to be used.
- If you use the IGNORE keyword, errors that occur while executing the INSERT statement are treated as warnings instead. For example, without IGNORE, a row that duplicates an existing UNIQUE index or PRIMARY KEY value in the table causes a duplicate-key error and the statement is aborted. With IGNORE, the row still is not inserted, but no error is issued. Data conversions that would trigger errors abort the statement if IGNORE is not specified. With IGNORE, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the mysql_info() C API function how many rows were actually inserted into the table.
- If you specify ON DUPLICATE KEY UPDATE, and a row is inserted that would cause a duplicate value in a UNIQUE index or PRIMARY KEY, an UPDATE of the old row is performed. See [Section 13.2.4.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

13.2.4.1. INSERT ... SELECT Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
  [INTO] tbl_name [(col_name, ...)]
  SELECT ...
  [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

With `INSERT ... SELECT`, you can quickly insert many rows into a table from one or many tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

The following conditions hold for a `INSERT ... SELECT` statements:

- Specify `IGNORE` to ignore rows that would cause duplicate-key violations.
- `DELAYED` is ignored with `INSERT ... SELECT`.
- The target table of the `INSERT` statement may appear in the `FROM` clause of the `SELECT` part of the query. (This was not possible in some older versions of MySQL.) In this case, MySQL creates a temporary table to hold the rows from the `SELECT` and then inserts those rows into the target table.
- `AUTO_INCREMENT` columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not allow concurrent inserts for `INSERT ... SELECT` statements.
- Currently, you cannot insert into a table and select from the same table in a subquery.

In the values part of `ON DUPLICATE KEY UPDATE`, you can refer to columns in other tables, as long as you do not use `GROUP BY` in the `SELECT` part. One side effect is that you must qualify non-unique column names in the values part.

13.2.4.2. `INSERT DELAYED` Syntax

```
INSERT DELAYED ...
```

The `DELAYED` option for the `INSERT` statement is a MySQL extension to standard SQL that is very useful if you have clients that cannot or need not wait for the `INSERT` to complete. This is a common situation when you use MySQL for logging and you also periodically run `SELECT` and `UPDATE` statements that take a long time to complete.

When a client uses `INSERT DELAYED`, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Another major benefit of using `INSERT DELAYED` is that inserts from many clients are bundled together and written in one block. This is much faster than performing many separate inserts.

Note that `INSERT DELAYED` is slower than a normal `INSERT` if the table is not otherwise in use. There is also the additional overhead for the server to handle a separate thread for each table for which there are delayed rows. This means that you should use `INSERT DELAYED` only when you are really sure that you need it.

The queued rows are held only in memory until they are inserted into the table. This means that if you terminate **mysqld** forcibly (for example, with `kill -9`) or if **mysqld** dies unexpectedly, *any queued rows that have not been written to disk are lost.*

There are some constraints on the use of `DELAYED`:

- `INSERT DELAYED` works only with MyISAM, MEMORY, and ARCHIVE tables. See [Section 14.1, “The MyISAM Storage Engine”](#), [Section 14.4, “The MEMORY \(HEAP\) Storage Engine”](#), and [Section 14.8, “The ARCHIVE Storage Engine”](#).

For MyISAM tables, if there are no free blocks in the middle of the data file, concurrent `SELECT` and `INSERT` statements are supported. Under these circumstances, you very seldom need to use `INSERT DELAYED` with MyISAM.

- `INSERT DELAYED` should be used only for `INSERT` statements that specify value lists. The server ignores `DELAYED` for `INSERT ... SELECT` or `INSERT ... ON DUPLICATE KEY UPDATE` statements.
- Because the `INSERT DELAYED` statement returns immediately, before the rows are inserted, you cannot use `LAST_INSERT_ID()` to get the `AUTO_INCREMENT` value that the statement might generate.
- `DELAYED` rows are not visible to `SELECT` statements until they actually have been inserted.
- `DELAYED` is ignored on slave replication servers because it could cause the slave to have different data than the master.

- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.

The following describes in detail what happens when you use the `DELAYED` option to `INSERT` or `REPLACE`. In this description, the “thread” is the thread that received an `INSERT DELAYED` statement and “handler” is the thread that handles all `INSERT DELAYED` statements for a particular table.

- When a thread executes a `DELAYED` statement for a table, a handler thread is created to process all `DELAYED` statements for the table, if no such handler already exists.
- The thread checks whether the handler has previously acquired a `DELAYED` lock; if not, it tells the handler thread to do so. The `DELAYED` lock can be obtained even if other threads have a `READ` or `WRITE` lock on the table. However, the handler waits for all `ALTER TABLE` locks or `FLUSH TABLES` statements to finish, to ensure that the table structure is up to date.
- The thread executes the `INSERT` statement, but instead of writing the row to the table, it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported to the client program.
- The client cannot obtain from the server the number of duplicate rows or the `AUTO_INCREMENT` value for the resulting row, because the `INSERT` returns before the insert operation has been completed. (If you use the C API, the `mysql_info()` function does not return anything meaningful, for the same reason.)
- The binary log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the binary log is updated when the first row is inserted.
- Each time that `delayed_insert_limit` rows are written, the handler checks whether any `SELECT` statements are still pending. If so, it allows these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new `INSERT DELAYED` statements are received within `delayed_insert_timeout` seconds, the handler terminates.

- If more than `delayed_queue_size` rows are pending in a specific handler queue, the thread requesting `INSERT DELAYED` waits until there is room in the queue. This is done to ensure that **mysqld** does not use all memory for the delayed memory queue.
- The handler thread shows up in the MySQL process list with `delayed_insert` in the Command column. It is killed if you execute a `FLUSH TABLES` statement or kill it with `KILL thread_id`. However, before exiting, it first stores all queued rows into the table. During this time it does not accept any new `INSERT` statements from other threads. If you execute an `INSERT DELAYED` statement after this, a new handler thread is created.

Note that this means that `INSERT DELAYED` statements have higher priority than normal `INSERT` statements if there is an `INSERT DELAYED` handler running. Other update statements have to wait until the `INSERT DELAYED` queue is empty, someone terminates the handler thread (with `KILL thread_id`), or someone executes a `FLUSH TABLES`.

- The following status variables provide information about `INSERT DELAYED` statements:

Status Variable	Meaning
<code>Delayed_insert_threads</code>	Number of handler threads
<code>Delayed_writes</code>	Number of rows written with <code>INSERT DELAYED</code>
<code>Not_flushed_delayed_rows</code>	Number of rows waiting to be written

You can view these variables by issuing a `SHOW STATUS` statement or by executing a **mysqladmin extended-status** command.

13.2.4.3. `INSERT ... ON DUPLICATE KEY UPDATE` Syntax

If you specify `ON DUPLICATE KEY UPDATE`, and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row is performed. For example, if column `a` is declared as `UNIQUE` and contains the value `1`, the following two statements have identical effect:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=c+1;
```

```
UPDATE table SET c=c+1 WHERE a=1;
```

The rows-affected value is 1 if the row is inserted as a new record and 2 if an existing record is updated.

If column *b* is also unique, the INSERT is equivalent to this UPDATE statement instead:

```
UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

If *a=1 OR b=2* matches several rows, only *one* row is updated. In general, you should try to avoid using an ON DUPLICATE KEY clause on tables with multiple unique indexes.

You can use the VALUES(*col_name*) function in the UPDATE clause to refer to column values from the INSERT portion of the INSERT ... UPDATE statement. In other words, VALUES(*col_name*) in the UPDATE clause refers to the value of *col_name* that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The VALUES() function is meaningful only in INSERT ... UPDATE statements and returns NULL otherwise. Example:

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
  ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=3;
INSERT INTO table (a,b,c) VALUES (4,5,6)
  ON DUPLICATE KEY UPDATE c=9;
```

The DELAYED option is ignored when you use ON DUPLICATE KEY UPDATE.

13.2.5. LOAD DATA INFILE Syntax

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
  [REPLACE | IGNORE]
  INTO TABLE tbl_name
  [FIELDS
    [TERMINATED BY 'string']
    [[OPTIONALLY] ENCLOSED BY 'char']
```

```

    [ESCAPED BY 'char']
]
[LINES
    [STARTING BY 'string']
    [TERMINATED BY 'string']
]
[IGNORE number LINES]
[(col_name_or_user_var, ...)]
[SET col_name = expr, ...)]

```

The LOAD DATA INFILE statement reads rows from a text file into a table at a very high speed. The filename must be given as a literal string.

LOAD DATA INFILE is the complement of SELECT ... INTO OUTFILE. (See [Section 13.2.7, “SELECT Syntax”](#).) To write data from a table to a file, use SELECT ... INTO OUTFILE. To read the file back into a table, use LOAD DATA INFILE. The syntax of the FIELDS and LINES clauses is the same for both statements. Both clauses are optional, but FIELDS must precede LINES if both are specified.

For more information about the efficiency of INSERT versus LOAD DATA INFILE and speeding up LOAD DATA INFILE, see [Section 7.2.16, “Speed of INSERT Statements”](#).

The character set indicated by the character_set_database system variable is used to interpret the information in the file. SET NAMES and the setting of character_set_client do not affect interpretation of input.

Note that it is currently not possible to load data files that use the ucs2 character set.

As of MySQL 5.0.19, the character_set_filesystem system variable controls the interpretation of the filename.

You can also load data files by using the **mysqlimport** utility; it operates by sending a LOAD DATA INFILE statement to the server. The --local option causes **mysqlimport** to read data files from the client host. You can specify the --compress option to get better performance over slow networks if the client and server support the compressed protocol. See [Section 8.14, “mysqlimport — A Data Import Program”](#).

If you use LOW_PRIORITY, execution of the LOAD DATA statement is delayed until no other clients are reading from the table.

If you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while `LOAD DATA` is executing. Using this option affects the performance of `LOAD DATA` a bit, even if no other thread is using the table at the same time.

The `LOCAL` keyword, if specified, is interpreted with respect to the client end of the connection:

- If `LOCAL` is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full pathname to specify its exact location. If given as a relative pathname, the name is interpreted relative to the directory in which the client program was started.
- If `LOCAL` is not specified, the file must be located on the server host and is read directly by the server. The server uses the following rules to locate the file:
 - If the filename is an absolute pathname, the server uses it as given.
 - If the filename is a relative pathname with one or more leading components, the server searches for the file relative to the server's data directory.
 - If a filename with no leading components is given, the server looks for the file in the database directory of the default database.

Note that, in the non-`LOCAL` case, these rules mean that a file named as `./myfile.txt` is read from the server's data directory, whereas the file named as `myfile.txt` is read from the database directory of the default database. For example, if `db1` is the default database, the following `LOAD DATA` statement reads the file `data.txt` from the database directory for `db1`, even though the statement explicitly loads the file into a table in the `db2` database:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Windows pathnames are specified using forward slashes rather than backslashes. If you do use backslashes, you must double them.

For security reasons, when reading text files located on the server, the files must

either reside in the database directory or be readable by all. Also, to use `LOAD DATA INFILE` on server files, you must have the `FILE` privilege. See [Section 5.8.3, “Privileges Provided by MySQL”](#).

Using `LOCAL` is a bit slower than letting the server access the files directly, because the contents of the file must be sent over the connection by the client to the server. On the other hand, you do not need the `FILE` privilege to load local files.

`LOCAL` works only if your server and your client both have been enabled to allow it. For example, if `mysqld` was started with `--local-infile=0`, `LOCAL` does not work. See [Section 5.7.4, “Security Issues with LOAD DATA LOCAL”](#).

On Unix, if you need `LOAD DATA` to read from a pipe, you can use the following technique (here we load the listing of the `/` directory into a table):

```
mkfifo /mysql/db/x/x
chmod 666 /mysql/db/x/x
find / -ls > /mysql/db/x/x
mysql -e "LOAD DATA INFILE 'x' INTO TABLE x" x
```

The `REPLACE` and `IGNORE` keywords control handling of input rows that duplicate existing rows on unique key values:

- If you specify `REPLACE`, input rows replace existing rows. In other words, rows that have the same value for a primary key or unique index as an existing row. See [Section 13.2.6, “REPLACE Syntax”](#).
- If you specify `IGNORE`, input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, the behavior depends on whether the `LOCAL` keyword is specified. Without `LOCAL`, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With `LOCAL`, the default behavior is the same as if `IGNORE` is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

If you want to ignore foreign key constraints during the load operation, you can issue a `SET FOREIGN_KEY_CHECKS=0` statement before executing `LOAD DATA`.

If you use `LOAD DATA INFILE` on an empty MyISAM table, all non-unique indexes are created in a separate batch (as for `REPAIR TABLE`). Normally, this makes `LOAD`

DATA INFILE much faster when you have many indexes. In some extreme cases, you can create the indexes even faster by turning them off with ALTER TABLE ... DISABLE KEYS before loading the file into the table and using ALTER TABLE ... ENABLE KEYS to re-create the indexes after loading the file. See [Section 7.2.16, “Speed of INSERT Statements”](#).

For both the LOAD DATA INFILE and SELECT ... INTO OUTFILE statements, the syntax of the FIELDS and LINES clauses is the same. Both clauses are optional, but FIELDS must precede LINES if both are specified.

If you specify a FIELDS clause, each of its subclauses (TERMINATED BY, [OPTIONALLY] ENCLOSED BY, and ESCAPED BY) is also optional, except that you must specify at least one of them.

If you specify no FIELDS clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

If you specify no LINES clause, the defaults are the same as if you had written this:

```
LINES TERMINATED BY '\n' STARTING BY ''
```

In other words, the defaults cause LOAD DATA INFILE to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip over any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret occurrences of tab, newline, or ‘\’ preceded by ‘\’ as literal characters that are part of field values.

Conversely, the defaults cause SELECT ... INTO OUTFILE to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.
- Use ‘\’ to escape instances of tab, newline, or ‘\’ that occur within field values.
- Write newlines at the ends of lines.

Backslash is the MySQL escape character within strings, so to write `FIELDS ESCAPED BY '\\'`, you must specify two backslashes for the value to be interpreted as a single backslash.

Note: If you have generated the text file on a Windows system, you might have to use `LINES TERMINATED BY '\\r\\n'` to read the file properly, because Windows programs typically use two characters as a line terminator. Some programs, such as **WordPad**, might use `\\r` as a line terminator when writing files. To read such files, use `LINES TERMINATED BY '\\r'`.

If all the lines you want to read in have a common prefix that you want to ignore, you can use `LINES STARTING BY 'prefix_string'` to skip over the prefix, *and anything before it*. If a line does not include the prefix, the entire line is skipped. Suppose that you issue the following statement:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
  FIELDS TERMINATED BY ',' LINES STARTING BY 'xxx';
```

If the data file looks like this:

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

The resulting rows will be `("abc",1)` and `("def",2)`. The third row in the file is skipped because it does not contain the prefix.

The `IGNORE number LINES` option can be used to ignore lines at the start of the file. For example, you can use `IGNORE 1 LINES` to skip over an initial header line containing column names:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA INFILE` to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, `LOAD DATA INFILE` will not interpret the contents of the file properly. Suppose that you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
SELECT * INTO OUTFILE 'data.txt'  
  FIELDS TERMINATED BY ','  
FROM table2;
```

To read the comma-delimited file back in, the correct statement would be:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2  
  FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown following, it wouldn't work because it instructs `LOAD DATA INFILE` to look for tabs between fields:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2  
  FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

`LOAD DATA INFILE` can be used to read files obtained from external sources. For example, many programs can export data in comma-separated values (CSV) format, such that lines have fields separated by commas and enclosed within double quotes. If lines in such a file are terminated by newlines, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name  
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'  
  LINES TERMINATED BY '\n';
```

Any of the field- or line-handling options can specify an empty string (''). If not empty, the `FIELDS [OPTIONALLY] ENCLOSED BY` and `FIELDS ESCAPED BY` values must be a single character. The `FIELDS TERMINATED BY`, `LINES STARTING BY`, and `LINES TERMINATED BY` values can be more than one character. For example, to write lines that are terminated by carriage return/linefeed pairs, or to read a file containing such lines, specify a `LINES TERMINATED BY '\r\n'` clause.

To read a file containing jokes that are separated by lines consisting of %, you can do this

```
CREATE TABLE jokes
  (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
  FIELDS TERMINATED BY ' '
  LINES TERMINATED BY '\n%\n' (joke);
```

FIELDS [OPTIONALLY] ENCLOSED BY controls quoting of fields. For output (SELECT ... INTO OUTFILE), if you omit the word OPTIONALLY, all fields are enclosed by the ENCLOSED BY character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify OPTIONALLY, the ENCLOSED BY character is used only to enclose values from columns that have a string data type (such as CHAR, BINARY, TEXT, or ENUM):

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Note that occurrences of the ENCLOSED BY character within a field value are escaped by prefixing them with the ESCAPED BY character. Also note that if you specify an empty ESCAPED BY value, it is possible to inadvertently generate output that cannot be read properly by LOAD DATA INFILE. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the ENCLOSED BY character, if present, is stripped from the ends of field values. (This is true regardless of whether OPTIONALLY is specified;

OPTIONALLY has no effect on input interpretation.) Occurrences of the ENCLOSED BY character preceded by the ESCAPED BY character are interpreted as part of the current field value.

If the field begins with the ENCLOSED BY character, instances of that character are recognized as terminating a field value only if followed by the field or line TERMINATED BY sequence. To avoid ambiguity, occurrences of the ENCLOSED BY character within a field value can be doubled and are interpreted as a single instance of the character. For example, if ENCLOSED BY ''' is specified, quotes are handled as shown here:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

FIELDS ESCAPED BY controls how to write or read special characters. If the FIELDS ESCAPED BY character is not empty, it is used to prefix the following characters on output:

- The FIELDS ESCAPED BY character
- The FIELDS [OPTIONALLY] ENCLOSED BY character
- The first character of the FIELDS TERMINATED BY and LINES TERMINATED BY values
- ASCII 0 (what is actually written following the escape character is ASCII '0', not a zero-valued byte)

If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

For input, if the FIELDS ESCAPED BY character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. The exceptions are an escaped '0' or 'N' (for example, \0 or \N if the escape character is '\'). These sequences are interpreted as ASCII NUL (a zero-valued byte) and NULL. The rules for NULL handling are described later in this section.

For more information about ‘\’-escape syntax, see [Section 9.1, “Literal Values”](#).

In certain cases, field- and line-handling options interact:

- If `LINES TERMINATED BY` is an empty string and `FIELDS TERMINATED BY` is non-empty, lines are also terminated with `FIELDS TERMINATED BY`.
- If the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values are both empty (' '), a fixed-row (non-delimited) format is used. With fixed-row format, no delimiters are used between fields (but you can still have a line terminator). Instead, column values are read and written using a field width wide enough to hold all values in the field. For `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, and `BIGINT`, the field widths are 4, 6, 8, 11, and 20, respectively, no matter what the declared display width is.

`LINES TERMINATED BY` is still used to separate lines. If a line does not contain all fields, the rest of the columns are set to their default values. If you do not have a line terminator, you should set this to ' '. In this case, the text file must contain all fields for each row.

Fixed-row format also affects handling of `NULL` values, as described later. Note that fixed-size format does not work if you are using a multi-byte character set.

Note: Before MySQL 5.0.6, fixed-row format used the display width of the column. For example, `INT(4)` was read or written using a field with a width of 4. However, if the column contained wider values, they were dumped to their full width, leading to the possibility of a “ragged” field holding values of different widths. Using a field wide enough to hold all values in the field prevents this problem. However, data files written before this change was made might not be reloaded correctly with `LOAD DATA INFILE` for MySQL 5.0.6 and up. This change also affects data files read by **mysqlimport** and written by **mysqldump --tab**, which use `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`.

Handling of `NULL` values varies according to the `FIELDS` and `LINES` options in use:

- For the default `FIELDS` and `LINES` values, `NULL` is written as a field value of `\N` for output, and a field value of `\N` is read as `NULL` for input (assuming

that the ESCAPED BY character is ‘\’).

- If FIELDS ENCLOSED BY is not empty, a field containing the literal word NULL as its value is read as a NULL value. This differs from the word NULL enclosed within FIELDS ENCLOSED BY characters, which is read as the string 'NULL'.
- If FIELDS ESCAPED BY is empty, NULL is written as the word NULL.
- With fixed-row format (which is used when FIELDS TERMINATED BY and FIELDS ENCLOSED BY are both empty), NULL is written as an empty string. Note that this causes both NULL values and empty strings in the table to be indistinguishable when written to the file because both are written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

An attempt to load NULL into a NOT NULL column causes assignment of the implicit default value for the column's data type and a warning, or an error in strict SQL mode. Implicit default values are discussed in [Section 11.1.4, “Data Type Default Values”](#).

Some cases are not supported by LOAD DATA INFILE:

- Fixed-size rows (FIELDS TERMINATED BY and FIELDS ENCLOSED BY both empty) and BLOB or TEXT columns.
- If you specify one separator that is the same as or a prefix of another, LOAD DATA INFILE cannot interpret the input properly. For example, the following FIELDS clause would cause problems:

```
FIELDS TERMINATED BY ''' ENCLOSED BY '''
```

- If FIELDS ESCAPED BY is empty, a field value that contains an occurrence of FIELDS ENCLOSED BY or LINES TERMINATED BY followed by the FIELDS TERMINATED BY value causes LOAD DATA INFILE to stop reading a field or line too early. This happens because LOAD DATA INFILE cannot properly determine where the field or line value ends.

The following example loads all columns of the persondata table:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

By default, when no column list is provided at the end of the `LOAD DATA INFILE` statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata (col1,col2, .
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match input fields with table columns.

Before MySQL 5.0.3, the column list must contain only names of columns in the table being loaded, and the `SET` clause is not supported. As of MySQL 5.0.3, the column list can contain either column names or user variables. With user variables, the `SET` clause enables you to perform transformations on their values before assigning the result to columns.

User variables in the `SET` clause can be used in several ways. The following example uses the first input column directly for the value of `t1.column1`, and assigns the second input column to a user variable that is subjected to a division operation before being used for the value of `t1.column2`:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

The `SET` clause can be used to supply values not derived from the input file. The following statement sets `column3` to the current date and time:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

You can also discard an input value by assigning it to a user variable and not assigning the variable to a table column:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @dummy, column2, @dummy, column3);
```

Use of the column/variable list and `SET` clause is subject to the following

restrictions:

- Assignments in the SET clause should have only column names on the left hand side of assignment operators.
- You can use subqueries in the right hand side of SET assignments. A subquery that returns a value to be assigned to a column may be a scalar subquery only. Also, you cannot use a subquery to select from the table that is being loaded.
- Lines ignored by an IGNORE clause are not processed for the column/variable list or SET clause.
- User variables cannot be used when loading data with fixed-row format because user variables do not have a display width.

When processing an input line, LOAD DATA splits it into fields and uses the values according to the column/variable list and the SET clause, if they are present. Then the resulting row is inserted into the table. If there are BEFORE INSERT or AFTER INSERT triggers for the table, they are activated before or after inserting the row, respectively.

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in [Section 11.1.4, “Data Type Default Values”](#).

An empty field value is interpreted differently than if the field value is missing:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to 0.
- For date and time types, the column is set to the appropriate “zero” value for the type. See [Section 11.3, “Date and Time Types”](#).

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an INSERT or UPDATE statement.

TIMESTAMP columns are set to the current date and time only if there is a NULL value for the column (that is, \N), or if the TIMESTAMP column's default value is the current timestamp and it is omitted from the field list when a field list is specified.

LOAD DATA INFILE regards all input as strings, so you cannot use numeric values for ENUM or SET columns the way you can with INSERT statements. All ENUM and SET values must be specified as strings.

BIT values cannot be loaded using binary notation (for example, b'011010'). To work around this, specify the values as regular integers and use the SET clause to convert them so that MySQL performs a numeric type conversion and loads them into the BIT column properly:

```
shell> cat /tmp/bit_test.txt
2
127
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
      -> INTO TABLE bit_test (@var1) SET b= CAST(@var1 AS SIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+-----+
| bin(b+0) |
+-----+
| 10      |
| 1111111 |
+-----+
2 rows in set (0.00 sec)
```

When the LOAD DATA INFILE statement finishes, it returns an information string in the following format:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

If you are using the C API, you can get information about the statement by calling the `mysql_info()` function. See [Section 22.2.3.34, “mysql_info\(\)”](#).

Warnings occur under the same circumstances as when values are inserted via the INSERT statement (see [Section 13.2.4, “INSERT Syntax”](#)), except that LOAD DATA INFILE also generates warnings when there are too few or too many fields

in the input row. The warnings are not stored anywhere; the number of warnings can be used only as an indication of whether everything went well.

You can use `SHOW WARNINGS` to get a list of the first `max_error_count` warnings as information about what went wrong. See [Section 13.5.4.25, “SHOW WARNINGS Syntax”](#).

13.2.6. REPLACE Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name [(col_name, ...)]
        VALUES ({expr | DEFAULT}, ...), (...), ...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name
        SET col_name={expr | DEFAULT}, ...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name [(col_name, ...)]
        SELECT ...
```

`REPLACE` works exactly like `INSERT`, except that if an old row in the table has the same value as a new row for a `PRIMARY KEY` or a `UNIQUE` index, the old row is deleted before the new row is inserted. See [Section 13.2.4, “INSERT Syntax”](#).

`REPLACE` is a MySQL extension to the SQL standard. It either inserts, or *deletes* and inserts. For another MySQL extension to standard SQL — that either inserts or *updates* — see [Section 13.2.4.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”](#).

Note that unless the table has a `PRIMARY KEY` or `UNIQUE` index, using a `REPLACE` statement makes no sense. It becomes equivalent to `INSERT`, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the `REPLACE` statement. Any missing columns are set to their default values, just as happens for `INSERT`. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as `SET col_name = col_name + 1`, the

reference to the column name on the right hand side is treated as `DEFAULT(col_name)`, so the assignment is equivalent to `SET col_name = DEFAULT(col_name) + 1`.

To use `REPLACE`, you must have both the `INSERT` and `DELETE` privileges for the table.

The `REPLACE` statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row `REPLACE`, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether `REPLACE` only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the `mysql_affected_rows()` function.

Currently, you cannot replace into a table and select from the same table in a subquery.

MySQL uses the following algorithm for `REPLACE` (and `LOAD DATA ... REPLACE`):

1. Try to insert the new row into the table
2. While the insertion fails because a duplicate-key error occurs for a primary key or unique index:
 1. Delete from the table the conflicting row that has the duplicate key value
 2. Try again to insert the new row into the table

13.2.7. SELECT Syntax

```

SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr, ...
  [FROM table_references]
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}
   [ASC | DESC], ... [WITH ROLLUP]]
  [HAVING where_condition]
  [ORDER BY {col_name | expr | position}
   [ASC | DESC], ...]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
  [PROCEDURE procedure_name(argument_list)]
  [INTO OUTFILE 'file_name' export_options
   | INTO DUMPFILE 'file_name'
   | INTO @var_name [, @var_name]]
  [FOR UPDATE | LOCK IN SHARE MODE]]

```

SELECT is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries. See [Section 13.2.7.2, “UNION Syntax”](#), and [Section 13.2.8, “Subquery Syntax”](#).

The most commonly used clauses of SELECT statements are these:

- Each *select_expr* indicates a column that you want to retrieve. There must be at least one *select_expr*.
- *table_references* indicates the table or tables from which to retrieve rows. Its syntax is described in [Section 13.2.7.1, “JOIN Syntax”](#).
- The WHERE clause, if given, indicates the condition or conditions that rows must satisfy to be selected. *where_condition* is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no WHERE clause.

In the WHERE clause, you can use any of the functions and operators that MySQL supports, except for aggregate (summary) functions. See [Chapter 12, *Functions and Operators*](#).

SELECT can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;  
-> 2
```

You are allowed to specify DUAL as a dummy table name in situations where no tables are referenced:

```
mysql> SELECT 1 + 1 FROM DUAL;  
-> 2
```

DUAL is purely for compatibility with some other database servers that require a FROM clause. MySQL does not require the clause if no tables are referenced.

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a HAVING clause must come after any GROUP BY clause and before any ORDER BY clause. The exception is that the INTO clause can appear either as shown in the syntax description or immediately preceding the FROM clause.

- A *select_expr* can be given an alias using AS *alias_name*. The alias is used as the expression's column name and can be used in GROUP BY, ORDER BY, or HAVING clauses. For example:

```
SELECT CONCAT(last_name, ', ', first_name) AS full_name  
FROM mytable ORDER BY full_name;
```

The AS keyword is optional when aliasing a *select_expr*. The preceding example could have been written like this:

```
SELECT CONCAT(last_name, ', ', first_name) full_name  
FROM mytable ORDER BY full_name;
```

However, because the AS is optional, a subtle problem can occur if you forget the comma between two *select_expr* expressions: MySQL interprets the second as an alias name. For example, in the following statement, *columnb* is treated as an alias name:

```
SELECT columna columnb FROM mytable;
```

For this reason, it is good practice to be in the habit of using AS explicitly when specifying column aliases.

- It is not allowable to use a column alias in a `WHERE` clause, because the column value might not yet be determined when the `WHERE` clause is executed. See [Section A.5.4, “Problems with Column Aliases”](#).
- The `FROM table_references` clause indicates the table or tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see [Section 13.2.7.1, “JOIN Syntax”](#). For each table specified, you can optionally specify an alias.

```
tbl_name [[AS] alias]
        [{USE|IGNORE|FORCE} INDEX (key_list)]
```

The use of `USE INDEX`, `IGNORE INDEX`, `FORCE INDEX` to give the optimizer hints about how to choose indexes is described in [Section 13.2.7.1, “JOIN Syntax”](#).

You can use `SET max_seeks_for_key=value` as an alternative way to force MySQL to prefer key scans instead of table scans. See [Section 5.2.2, “Server System Variables”](#).

- You can refer to a table within the default database as *tbl_name*, or as *db_name.tbl_name* to specify a database explicitly. You can refer to a column as *col_name*, *tbl_name.col_name*, or *db_name.tbl_name.col_name*. You need not specify a *tbl_name* or *db_name.tbl_name* prefix for a column reference unless the reference would be ambiguous. See [Section 9.2.1, “Identifier Qualifiers”](#), for examples of ambiguity that require the more explicit column reference forms.
- A table reference can be aliased using `tbl_name AS alias_name` or `tbl_name alias_name`:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
        WHERE t1.name = t2.name;
```

```
SELECT t1.name, t2.salary FROM employee t1, info t2
        WHERE t1.name = t2.name;
```

- Columns selected for output can be referred to in `ORDER BY` and `GROUP BY` clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

```
SELECT college, region, seed FROM tournament
```

```
ORDER BY region, seed;
```

```
SELECT college, region AS r, seed AS s FROM tournament  
ORDER BY r, s;
```

```
SELECT college, region, seed FROM tournament  
ORDER BY 2, 3;
```

To sort in reverse order, add the `DESC` (descending) keyword to the name of the column in the `ORDER BY` clause that you are sorting by. The default is ascending order; this can be specified explicitly using the `ASC` keyword.

Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- If you use `GROUP BY`, output rows are sorted according to the `GROUP BY` columns as if you had an `ORDER BY` for the same columns. To avoid the overhead of sorting that `GROUP BY` produces, add `ORDER BY NULL`:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

- MySQL extends the `GROUP BY` clause so that you can also specify `ASC` and `DESC` after columns named in the clause:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;
```

- MySQL extends the use of `GROUP BY` to allow selecting fields that are not mentioned in the `GROUP BY` clause. If you are not getting the results that you expect from your query, please read the description of `GROUP BY` found in [Section 12.10, “Functions and Modifiers for Use with GROUP BY Clauses”](#).
- `GROUP BY` allows a `WITH ROLLUP` modifier. See [Section 12.10.2, “GROUP BY Modifiers”](#).
- The `HAVING` clause is applied nearly last, just before items are sent to the client, with no optimization. (`LIMIT` is applied after `HAVING`.)

A `HAVING` clause can refer to any column or alias named in a *select_expr* in the `SELECT` list or in outer subqueries, and to aggregate functions. However, the SQL standard requires that `HAVING` must reference only columns in the `GROUP BY` clause or columns used in aggregate functions. To accommodate both standard SQL and the MySQL-specific behavior of

being able to refer columns in the `SELECT` list, MySQL 5.0.2 and up allows `HAVING` to refer to columns in the `SELECT` list, columns in the `GROUP BY` clause, columns in outer subqueries, and to aggregate functions.

For example, the following statement works in MySQL 5.0.2 but produces an error for earlier versions:

```
mysql> SELECT COUNT(*) FROM t GROUP BY col1 HAVING col1 = 2;
```

If the `HAVING` clause refers to a column that is ambiguous, a warning occurs. In the following statement, `col2` is ambiguous because it is used as both an alias and a column name:

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

Preference is given to standard SQL behavior, so if a `HAVING` column name is used both in `GROUP BY` and as an aliased column in the output column list, preference is given to the column in the `GROUP BY` column.

- Do not use `HAVING` for items that should be in the `WHERE` clause. For example, do not write the following:

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

Write this instead:

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- The `HAVING` clause can refer to aggregate functions, which the `WHERE` clause cannot:

```
SELECT user, MAX(salary) FROM users  
GROUP BY user HAVING MAX(salary) > 10;
```

(This did not work in some older versions of MySQL.)

- MySQL allows duplicate column names. That is, there can be more than one `select_expr` with the same name. This is an extension to standard SQL. Because MySQL also allows `GROUP BY` and `HAVING` to refer to `select_expr` values, this can result in an ambiguity:

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

In that statement, both columns have the name a. To ensure that the correct column is used for grouping, use different names for each *select_expr*.

- When MySQL resolves an unqualified column or alias reference in an ORDER BY, GROUP BY, or HAVING clause, it first searches for the name in the *select_expr* values. If the name is not found, it looks in the columns of the tables named in the FROM clause.
- The LIMIT clause can be used to constrain the number of rows returned by the SELECT statement. LIMIT takes one or two numeric arguments, which must both be non-negative integer constants (except when using prepared statements).

With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

```
SELECT * FROM tbl LIMIT 5,10; # Retrieve rows 6-15
```

To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

With one argument, the value specifies the number of rows to return from the beginning of the result set:

```
SELECT * FROM tbl LIMIT 5; # Retrieve first 5 rows
```

In other words, LIMIT row_count is equivalent to LIMIT 0, row_count.

For prepared statements, you can use placeholders (supported as of MySQL version 5.0.7). The following statements will return one row from the tbl table:

```
SET @a=1;  
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';  
EXECUTE STMT USING @a;
```

The following statements will return the second to sixth row from the tbl table:

```
SET @skip=1; SET @numrows=5;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
EXECUTE STMT USING @skip, @numrows;
```

For compatibility with PostgreSQL, MySQL also supports the `LIMIT row_count OFFSET offset` syntax.

- The `SELECT ... INTO OUTFILE 'file_name'` form of `SELECT` writes the selected rows to a file. The file is created on the server host, so you must have the `FILE` privilege to use this syntax. *file_name* cannot be an existing file, which among other things prevents files such as `/etc/passwd` and database tables from being destroyed. As of MySQL 5.0.19, the `character_set_filesystem` system variable controls the interpretation of the filename.

The `SELECT ... INTO OUTFILE` statement is intended primarily to let you very quickly dump a table to a text file on the server machine. If you want to create the resulting file on some client host other than the server host, you cannot use `SELECT ... INTO OUTFILE`. In that case, you should instead use a command such as `mysql -e "SELECT ..." > file_name` to generate the file on the client host.

`SELECT ... INTO OUTFILE` is the complement of `LOAD DATA INFILE`; the syntax for the *export_options* part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA INFILE` statement. See [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).

`FIELDS ESCAPED BY` controls how to write special characters. If the `FIELDS ESCAPED BY` character is not empty, it is used as a prefix that precedes following characters on output:

- The `FIELDS ESCAPED BY` character
- The `FIELDS [OPTIONALLY] ENCLOSED BY` character
- The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
- ASCII NUL (the zero-valued byte; what is actually written following the escape character is ASCII ‘0’, not a zero-valued byte)

The `FIELDS TERMINATED BY`, `ENCLOSED BY`, `ESCAPED BY`, or `LINES TERMINATED BY` characters *must* be escaped so that you can read the file back in reliably. ASCII NUL is escaped to make it easier to view with some pagers.

The resulting file does not have to conform to SQL syntax, so nothing else need be escaped.

If the `FIELDS ESCAPED BY` character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

Here is an example that produces a file in the comma-separated values (CSV) format used by many programs:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'  
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ''''  
  LINES TERMINATED BY '\n'  
FROM test_table;
```

- If you use `INTO DUMPFILE` instead of `INTO OUTFILE`, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a BLOB value in a file.
- The `INTO` clause can name a list of one or more user-defined variables. The selected values are assigned to the variables. The number of variables must match the number of columns.

Within a stored routine, the variables can be routine parameters or local variables. See [Section 17.2.7.3, “SELECT . . . INTO Statement”](#).

- **Note:** Any file created by `INTO OUTFILE` or `INTO DUMPFILE` is writable by all users on the server host. The reason for this is that the MySQL server cannot create a file that is owned by anyone other than the user under whose account it is running. (You should *never* run **mysqld** as root for this and other reasons.) The file thus must be world-writable so that you can manipulate its contents.
- The `SELECT` syntax description at the beginning this section shows the `INTO`

clause near the end of the statement. It is also possible to use `INTO OUTFILE` or `INTO DUMPFILE` immediately preceding the `FROM` clause.

- A `PROCEDURE` clause names a procedure that should process the data in the result set. For an example, see [Section 24.3.1, “Procedure Analyse”](#).
- If you use `FOR UPDATE` with a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction. Using `LOCK IN SHARE MODE` sets a shared lock that allows other transactions to read the examined rows but not to update or delete them. See [Section 14.2.10.5, “SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE Locking Reads”](#).

Following the `SELECT` keyword, you can use a number of options that affect the operation of the statement.

The `ALL`, `DISTINCT`, and `DISTINCTROW` options specify whether duplicate rows should be returned. If none of these options are given, the default is `ALL` (all matching rows are returned). `DISTINCT` and `DISTINCTROW` are synonyms and specify removal of duplicate rows from the result set.

`HIGH_PRIORITY`, `STRAIGHT_JOIN`, and options beginning with `SQL_` are MySQL extensions to standard SQL.

- `HIGH_PRIORITY` gives the `SELECT` higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A `SELECT HIGH_PRIORITY` query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free.

`HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`.

- `STRAIGHT_JOIN` forces the optimizer to join the tables in the order in which they are listed in the `FROM` clause. You can use this to speed up a query if the optimizer joins the tables in non-optimal order. See [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#). `STRAIGHT_JOIN` also can be used in the `table_references` list. See [Section 13.2.7.1, “JOIN Syntax”](#).
- `SQL_BIG_RESULT` can be used with `GROUP BY` or `DISTINCT` to tell the

optimizer that the result set has many rows. In this case, MySQL directly uses disk-based temporary tables if needed, and prefers sorting to using a temporary table with a key on the `GROUP BY` elements.

- `SQL_BUFFER_RESULT` forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client.
- `SQL_SMALL_RESULT` can be used with `GROUP BY` or `DISTINCT` to tell the optimizer that the result set is small. In this case, MySQL uses fast temporary tables to store the resulting table instead of using sorting. This should not normally be needed.
- `SQL_CALC_FOUND_ROWS` tells MySQL to calculate how many rows there would be in the result set, disregarding any `LIMIT` clause. The number of rows can then be retrieved with `SELECT FOUND_ROWS()`. See [Section 12.9.3, “Information Functions”](#).
- `SQL_CACHE` tells MySQL to store the query result in the query cache if you are using a `query_cache_type` value of 2 or `DEMAND`. For a query that uses `UNION` or subqueries, this option effects any `SELECT` in the query. See [Section 5.14, “The MySQL Query Cache”](#).
- `SQL_NO_CACHE` tells MySQL not to store the query result in the query cache. See [Section 5.14, “The MySQL Query Cache”](#). For a query that uses `UNION` or subqueries, this option effects any `SELECT` in the query.

13.2.7.1. JOIN Syntax

MySQL supports the following `JOIN` syntaxes for the *table_references* part of `SELECT` statements and multiple-table `DELETE` and `UPDATE` statements:

```
table_references:  
    table_reference [, table_reference] ...
```

```
table_reference:  
    table_factor  
    | join_table
```

```
table_factor:  
    tbl_name [[AS] alias]
```

```

    [{USE|IGNORE|FORCE} INDEX (key_list)]
| ( table_references )
| { OJ table_reference LEFT OUTER JOIN table_reference
    ON conditional_expr }

```

join_table:

```

    table_reference [INNER | CROSS] JOIN table_factor [join_conditio
| table_reference STRAIGHT_JOIN table_factor
| table_reference STRAIGHT_JOIN table_factor ON condition
| table_reference LEFT [OUTER] JOIN table_reference join_condition
| table_reference NATURAL [LEFT [OUTER]] JOIN table_factor
| table_reference RIGHT [OUTER] JOIN table_reference join_conditio
| table_reference NATURAL [RIGHT [OUTER]] JOIN table_factor

```

join_condition:

```

    ON conditional_expr
| USING (column_list)

```

A table reference is also known as a join expression.

The syntax of *table_factor* is extended in comparison with the SQL Standard. The latter accepts only *table_reference*, not a list of them inside a pair of parentheses.

This is a conservative extension if we consider each comma in a list of *table_reference* items as equivalent to an inner join. For example:

```

SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
    ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

```

is equivalent to:

```

SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
    ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)

```

In MySQL, CROSS JOIN is a syntactic equivalent to INNER JOIN (they can replace each other). In standard SQL, they are not equivalent. INNER JOIN is used with an ON clause, CROSS JOIN is used otherwise.

In versions of MySQL prior to 5.0.1, parentheses in *table_references* were just omitted and all join operations were grouped to the left. In general, parentheses can be ignored in join expressions containing only inner join operations. As of 5.0.1, nested joins are allowed (see [Section 7.2.10, “Nested Join Optimization”](#)).

Further changes in join processing were made in 5.0.12 to make MySQL more compliant with standard SQL. These changes are described later in this section.

The following list describes general factors to take into account when writing joins.

- A table reference can be aliased using `tbl_name AS alias_name` or `tbl_name alias_name`:

```
SELECT t1.name, t2.salary
   FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name
```

```
SELECT t1.name, t2.salary
   FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;
```

- `INNER JOIN` and `,` (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table is joined to each and every row in the second table).

However, the precedence of the comma operator is less than that of `INNER JOIN`, `CROSS JOIN`, `LEFT JOIN`, and so on. If you mix comma joins with the other join types when there is a join condition, an error of the form `Unknown column 'col_name' in 'on clause'` may occur. Information about dealing with this problem is given later in this section.

- The `ON` conditional is any conditional expression of the form that can be used in a `WHERE` clause. Generally, you should use the `ON` clause for conditions that specify how to join tables, and the `WHERE` clause to restrict which rows you want in the result set.
- If there is no matching row for the right table in the `ON` or `USING` part in a `LEFT JOIN`, a row with all columns set to `NULL` is used for the right table. You can use this fact to find rows in a table that have no counterpart in another table:

```
SELECT table1.* FROM table1
   LEFT JOIN table2 ON table1.id=table2.id
   WHERE table2.id IS NULL;
```

This example finds all rows in `table1` with an `id` value that is not present in `table2` (that is, all rows in `table1` with no corresponding row in `table2`).

This assumes that `table2.id` is declared `NOT NULL`. See [Section 7.2.9, “LEFT JOIN and RIGHT JOIN Optimization”](#).

- The `USING(column_list)` clause names a list of columns that must exist in both tables. If tables `a` and `b` both contain columns `c1`, `c2`, and `c3`, the following join compares corresponding columns from the two tables:

```
a LEFT JOIN b USING (c1, c2, c3)
```
- The `NATURAL [LEFT] JOIN` of two tables is defined to be semantically equivalent to an `INNER JOIN` or a `LEFT JOIN` with a `USING` clause that names all columns that exist in both tables.
- `RIGHT JOIN` works analogously to `LEFT JOIN`. To keep code portable across databases, it is recommended that you use `LEFT JOIN` instead of `RIGHT JOIN`.
- The `{ OJ ... LEFT OUTER JOIN ... }` syntax shown in the join syntax description exists only for compatibility with ODBC. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.
- `STRAIGHT_JOIN` is identical to `JOIN`, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order.

Some join examples:

```
SELECT * FROM table1, table2;
```

```
SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id;
```

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;
```

```
SELECT * FROM table1 LEFT JOIN table2 USING (id);
```

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id  
LEFT JOIN table3 ON table2.id=table3.id;
```

You can provide hints as to which index MySQL should use when retrieving information from a table. By specifying `USE INDEX (key_list)`, you can tell MySQL to use only one of the possible indexes to find rows in the table. The

alternative syntax `IGNORE INDEX (key_list)` can be used to tell MySQL to not use some particular index. These hints are useful if `EXPLAIN` shows that MySQL is using the wrong index from the list of possible indexes.

You can also use `FORCE INDEX`, which acts like `USE INDEX (key_list)` but with the addition that a table scan is assumed to be *very* expensive. In other words, a table scan is used only if there is no way to use one of the given indexes to find rows in the table.

`USE INDEX`, `IGNORE INDEX`, and `FORCE INDEX` affect only which indexes are used when MySQL decides how to find rows in the table and how to do the join. They do not affect whether an index is used when resolving an `ORDER BY` or `GROUP BY`.

`USE KEY`, `IGNORE KEY`, and `FORCE KEY` are synonyms for `USE INDEX`, `IGNORE INDEX`, and `FORCE INDEX`.

Examples:

```
SELECT * FROM table1 USE INDEX (key1,key2)
  WHERE key1=1 AND key2=2 AND key3=3;
```

```
SELECT * FROM table1 IGNORE INDEX (key3)
  WHERE key1=1 AND key2=2 AND key3=3;
```

Join Processing Changes in MySQL 5.0.12

Beginning with MySQL 5.0.12, natural joins and joins with `USING`, including outer join variants, are processed according to the SQL:2003 standard. The goal was to align the syntax and semantics of MySQL with respect to `NATURAL JOIN` and `JOIN . . . USING` according to SQL:2003. However, these changes in join processing can result in different output columns for some joins. Also, some queries that appeared to work correctly in older versions must be rewritten to comply with the standard.

These changes have five main aspects:

- The way that MySQL determines the result columns of `NATURAL` or `USING` join operations (and thus the result of the entire `FROM` clause).
- Expansion of `SELECT *` and `SELECT tbl_name.*` into a list of selected columns.

- Resolution of column names in NATURAL or USING joins.
- Transformation of NATURAL or USING joins into JOIN ... ON.
- Resolution of column names in the ON condition of a JOIN ... ON.

The following list provides more detail about several effects of the 5.0.12 change in join processing. The term “previously” means “prior to MySQL 5.0.12.”

- The columns of a NATURAL join or a USING join may be different from previously. Specifically, redundant output columns no longer appear, and the order of columns for SELECT * expansion may be different from before.

Consider this set of statements:

```
CREATE TABLE t1 (i INT, j INT);
CREATE TABLE t2 (k INT, j INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 JOIN t2 USING (j);
```

Previously, the statements produced this output:

```
+-----+-----+-----+-----+
| i     | j     | k     | j     |
+-----+-----+-----+-----+
| 1     | 1     | 1     | 1     |
+-----+-----+-----+-----+
| i     | j     | k     | j     |
+-----+-----+-----+-----+
| 1     | 1     | 1     | 1     |
+-----+-----+-----+-----+
```

In the first SELECT statement, column j appears in both tables and thus becomes a join column, so, according to standard SQL, it should appear only once in the output, not twice. Similarly, in the second SELECT statement, column j is named in the USING clause and should appear only once in the output, not twice. But in both cases, the redundant column is not eliminated. Also, the order of the columns is not correct according to standard SQL.

Now the statements produce this output:

```
+-----+-----+-----+
| j      | i      | k      |
+-----+-----+-----+
|      1 |      1 |      1 |
+-----+-----+-----+
+-----+-----+-----+
| j      | i      | k      |
+-----+-----+-----+
|      1 |      1 |      1 |
+-----+-----+-----+
```

The redundant column is eliminated and the column order is correct according to standard SQL:

- First, coalesced common columns of the two joined tables, in the order in which they occur in the first table
- Second, columns unique to the first table, in order in which they occur in that table
- Third, columns unique to the second table, in order in which they occur in that table

The single result column that replaces two common columns is defined via the coalesce operation. That is, from two $t1.a$ and $t2.a$ the resulting single join column a is defined as $a = \text{COALESCE}(t1.a, t2.a)$, where:

$\text{COALESCE}(x, y) = (\text{CASE WHEN } V1 \text{ IS NOT NULL THEN } V1 \text{ ELSE } V2 \text{ END})$

If the join operation is any other join, the result columns of the join consists of the concatenation of all columns of the joined tables. This is the same as previously.

A consequence of the definition of coalesced columns is that, for outer joins, the coalesced column contains the value of the non-NULL column if one of the two columns is always NULL. If neither or both columns are NULL, both common columns have the same value, so it doesn't matter which one is chosen as the value of the coalesced column. A simple way to interpret this is to consider that a coalesced column of an outer join is represented by the common column of the inner table of a JOIN. Suppose that the tables

t1(a,b) and t2(a,c) have the following contents:

t1	t2
1 x	2 z
2 y	3 w

Then:

```
mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
+-----+-----+-----+
| a     | b     | c     |
+-----+-----+-----+
| 1     | x     | NULL  |
| 2     | y     | y     |
+-----+-----+-----+
```

Here column a contains the values of t1.a.

```
mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
+-----+-----+-----+
| a     | c     | b     |
+-----+-----+-----+
| 2     | y     | y     |
| 3     | z     | NULL  |
+-----+-----+-----+
```

Here column a contains the values of t2.a.

Compare these results to the otherwise equivalent queries with JOIN ... ON:

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
+-----+-----+-----+-----+
| a     | b     | a     | c     |
+-----+-----+-----+-----+
| 1     | x     | NULL  | NULL  |
| 2     | y     | 2     | y     |
+-----+-----+-----+-----+
```

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
+-----+-----+-----+-----+
| a     | b     | a     | c     |
+-----+-----+-----+-----+
| 2     | y     | 2     | y     |
| NULL  | NULL  | 3     | z     |
+-----+-----+-----+-----+
```

- Previously, a USING clause could be rewritten as an ON clause that compares corresponding columns. For example, the following two clauses were semantically identical:

```
a LEFT JOIN b USING (c1, c2, c3)
a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
```

Now the two clauses no longer are quite the same:

- With respect to determining which rows satisfy the join condition, both joins remain semantically identical.
- With respect to determining which columns to display for SELECT * expansion, the two joins are not semantically identical. The USING join selects the coalesced value of corresponding columns, whereas the ON join selects all columns from all tables. For the preceding USING join, SELECT * selects these values:

```
COALESCE(a.c1, b.c1), COALESCE(a.c2, b.c2), COALESCE(a.c3, b.c3)
```

For the ON join, SELECT * selects these values:

```
a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
```

With an inner join, COALESCE(a.c1, b.c1) is the same as either a.c1 or b.c1 because both columns will have the same value. With an outer join (such as LEFT JOIN), one of the two columns can be NULL. That column will be omitted from the result.

- The evaluation of multi-way natural joins differs in a very important way that affects the result of NATURAL or USING joins and that can require query rewriting. Suppose that you have three tables t1(a, b), t2(c, b), and t3(a, c) that each have one row: t1(1, 2), t2(10, 2), and t3(7, 10). Suppose also that you have this NATURAL JOIN on the three tables:

```
SELECT ... FROM t1 NATURAL JOIN t2 NATURAL JOIN t3;
```

Previously, the left operand of the second join was considered to be t2, whereas it should be the nested join (t1 NATURAL JOIN t2). As a result, the columns of t3 are checked for common columns only in t2, and, if t3 has common columns with t1, these columns are not used as equi-join

columns. Thus, previously, the preceding query was transformed to the following equi-join:

```
SELECT ... FROM t1, t2, t3
  WHERE t1.b = t2.b AND t2.c = t3.c;
```

That join is missing one more equi-join predicate ($t1.a = t3.a$). As a result, it produces one row, not the empty result that it should. The correct equivalent query is this:

```
SELECT ... FROM t1, t2, t3
  WHERE t1.b = t2.b AND t2.c = t3.c AND t1.a = t3.a;
```

If you require the same query result in current versions of MySQL as in older versions, rewrite the natural join as the first equi-join.

- Previously, the comma operator (,) and JOIN both had the same precedence, so the join expression $t1, t2 \text{ JOIN } t3$ was interpreted as $((t1, t2) \text{ JOIN } t3)$. Now JOIN has higher precedence, so the expression is interpreted as $(t1, (t2 \text{ JOIN } t3))$. This change affects statements that use an ON clause, because that clause can refer only to columns in the operands of the join, and the change in precedence changes interpretation of what those operands are.

Example:

```
CREATE TABLE t1 (i1 INT, j1 INT);
CREATE TABLE t2 (i2 INT, j2 INT);
CREATE TABLE t3 (i3 INT, j3 INT);
INSERT INTO t1 VALUES(1,1);
INSERT INTO t2 VALUES(1,1);
INSERT INTO t3 VALUES(1,1);
SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
```

Previously, the SELECT was legal due to the implicit grouping of $t1, t2$ as $(t1, t2)$. Now the JOIN takes precedence, so the operands for the ON clause are $t2$ and $t3$. Because $t1.i1$ is not a column in either of the operands, the result is an Unknown column 't1.i1' in 'on clause' error. To allow the join to be processed, group the first two tables explicitly with parentheses so that the operands for the ON clause are $(t1, t2)$ and $t3$:

```
SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
```

Alternatively, avoid the use of the comma operator and use JOIN instead:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
```

This change also applies to statements that mix the comma operator with INNER JOIN, CROSS JOIN, LEFT JOIN, and RIGHT JOIN, all of which now have higher precedence than the comma operator.

- Previously, the ON clause could refer to columns in tables named to its right. Now an ON clause can refer only to its operands.

Example:

```
CREATE TABLE t1 (i1 INT);  
CREATE TABLE t2 (i2 INT);  
CREATE TABLE t3 (i3 INT);  
SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
```

Previously, the SELECT statement was legal. Now the statement fails with an Unknown column 'i3' in 'on clause' error because i3 is a column in t3, which is not an operand of the ON clause. The statement should be rewritten as follows:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
```

- Resolution of column names in NATURAL or USING joins is different than previously. For column names that are outside the FROM clause, MySQL now handles a superset of the queries compared to previously. That is, in cases when MySQL formerly issued an error that some column is ambiguous, the query now is handled correctly. This is due to the fact that MySQL now treats the common columns of NATURAL or USING joins as a single column, so when a query refers to such columns, the query compiler does not consider them as ambiguous.

Example:

```
SELECT * FROM t1 NATURAL JOIN t2 WHERE b > 1;
```

Previously, this query would produce an error ERROR 1052 (23000): Column 'b' in where clause is ambiguous. Now the query produces the correct result:

```

+-----+-----+-----+
| b     | c     | y     |
+-----+-----+-----+
|      4 |      2 |      3 |
+-----+-----+-----+

```

One extension of MySQL compared to the SQL:2003 standard is that MySQL allows you to qualify the common (coalesced) columns of NATURAL or USING joins (just as previously), while the standard disallows that.

13.2.7.2. UNION Syntax

```

SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]

```

UNION is used to combine the result from multiple SELECT statements into a single result set.

The column names from the first SELECT statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each SELECT statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If the data types of corresponding SELECT columns do not match, the types and lengths of the columns in the UNION result take into account the values retrieved by all of the SELECT statements. For example, consider the following:

```

mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+-----+
| REPEAT('a',1) |
+-----+
| a              |
| bbbbbbbbbbb   |
+-----+

```

(In some earlier versions of MySQL, only the type and length from the first SELECT would have been used and the second row would have been truncated to a length of 1.)

The SELECT statements are normal select statements, but with the following

restrictions:

- Only the last SELECT statement can use INTO OUTFILE.
- HIGH_PRIORITY cannot be used with SELECT statements that are part of a UNION. If you specify it for the first SELECT, it has no effect. If you specify it for any subsequent SELECT statements, a syntax error results.

The default behavior for UNION is that duplicate rows are removed from the result. The optional DISTINCT keyword has no effect other than the default because it also specifies duplicate-row removal. With the optional ALL keyword, duplicate-row removal does not occur and the result includes all matching rows from all the SELECT statements.

You can mix UNION ALL and UNION DISTINCT in the same query. Mixed UNION types are treated such that a DISTINCT union overrides any ALL union to its left. A DISTINCT union can be produced explicitly by using UNION DISTINCT or implicitly by using UNION with no following DISTINCT or ALL keyword.

To use an ORDER BY or LIMIT clause to sort or limit the entire UNION result, parenthesize the individual SELECT statements and place the ORDER BY or LIMIT after the last one. The following example uses both clauses:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

This kind of ORDER BY cannot use column references that include a table name (that is, names in *tbl_name.col_name* format). Instead, provide a column alias in the first SELECT statement and refer to the alias in the ORDER BY. (Alternatively, refer to the column in the ORDER BY using its column position. However, use of column positions is deprecated.)

Also, if a column to be sorted is aliased, the ORDER BY clause *must* refer to the alias, not the column name. The first of the following statements will work, but the second will fail with an Unknown column 'a' in 'order clause' error:

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

To apply `ORDER BY` or `LIMIT` to an individual `SELECT`, place the clause inside the parentheses that enclose the `SELECT`:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

Use of `ORDER BY` for individual `SELECT` statements implies nothing about the order in which the rows appear in the final result because `UNION` by default produces an unordered set of rows. If `ORDER BY` appears with `LIMIT`, it is used to determine the subset of the selected rows to retrieve for the `SELECT`, but does not necessarily affect the order of those rows in the final `UNION` result. If `ORDER BY` appears without `LIMIT` in a `SELECT`, it is optimized away because it will have no effect anyway.

To cause rows in a `UNION` result to consist of the sets of rows retrieved by each `SELECT` one after the other, select an additional column in each `SELECT` to use as a sort column and add an `ORDER BY` following the last `SELECT`:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

To additionally maintain sort order within individual `SELECT` results, add a secondary column to the `ORDER BY` clause:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

13.2.8. Subquery Syntax

A subquery is a `SELECT` statement within another statement.

Starting with MySQL 4.1, all subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, `SELECT * FROM t1 ...` is the *outer query* (or *outer statement*), and `(SELECT column1 FROM t2)` is the *subquery*. We say that the subquery is *nested* within the outer query, and in fact it is possible to nest subqueries within other subqueries, to a considerable depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- They are, in many people's opinion, readable. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
(SELECT COUNT(*) /* no hint */ FROM t2
WHERE NOT EXISTS
(SELECT * FROM t3
WHERE ROW(5*t2.s1,77)=
(SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
(SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used. A subquery can contain any of the keywords or clauses that an ordinary `SELECT` can contain: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, index hints, `UNION` constructs, comments, functions, and so on.

One restriction is that a subquery's outer statement must be one of: `SELECT`,

INSERT, UPDATE, DELETE, SET, or DO. Another restriction is that currently you cannot modify a table and select from the same table in a subquery. This applies to statements such as DELETE, INSERT, REPLACE, UPDATE, and (because subqueries can be used in the SET clause) LOAD DATA INFILE.

A more comprehensive discussion of restrictions on subquery use, including performance issues for certain forms of subquery syntax, is given in [Section I.3, “Restrictions on Subqueries”](#).

13.2.8.1. The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication whether it can be NULL, and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

The subquery in this SELECT returns a single value ('abcde') that has a data type of CHAR, a length of 5, a character set and collation equal to the defaults in effect at CREATE TABLE time, and an indication that the value in the column can be NULL. In fact, almost all subqueries can be NULL. If the table used in the example were empty, the value of the subquery would be NULL.

There are a few contexts in which a scalar subquery cannot be used. If a statement allows only a literal value, you cannot use a subquery. For example, LIMIT requires literal integer arguments, and LOAD DATA INFILE requires a literal string filename. You cannot use subqueries to supply these values.

When you see examples in the following sections that contain the rather spartan construct (SELECT column1 FROM t1), imagine that your own code contains much more diverse and complex constructions.

Suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
```

```
INSERT INTO t2 VALUES (2);
```

Then perform a SELECT:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is 2 because there is a row in t2 containing a column s1 that has a value of 2.

A scalar subquery can be part of an expression, but remember the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.2.8.2. Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where *comparison_operator* is one of these operators:

```
= > < >= <= <>
```

For example:

```
... 'a' = (SELECT column1 FROM t1)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the values in table t1 that are equal to a maximum value in table t2:

```
SELECT column1 FROM t1  
WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one of the tables. It finds all rows in table t1 containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison performed with one of these operators, the subquery must return a scalar, with the exception that = can be used with row subqueries. See [Section 13.2.8.5, “Row Subqueries”](#).

13.2.8.3. Subqueries with ANY, IN, and SOME

Syntax:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

The ANY keyword, which must follow a comparison operator, means “return TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table t1 containing (10). The expression is TRUE if table t2 contains (21, 14, 7) because there is a value 7 in t2 that is less than 10. The expression is FALSE if table t2 contains (20, 10), or if table t2 is empty. The expression is UNKNOWN if table t2 contains (NULL, NULL, NULL).

The word IN is an alias for = ANY. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

However, NOT IN is not an alias for <> ANY, but for <> ALL. See [Section 13.2.8.4, “Subqueries with ALL”](#).

The word SOME is an alias for ANY. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word SOME is rare, but this example shows why it might be useful. To most people's ears, the English phrase “a is not equal to any b” means “there is no b which is equal to a,” but that is not what is meant by the SQL syntax. The

syntax means “there is some b to which a is not equal.” Using `<> SOME` instead helps ensure that everyone understands the true meaning of the query.

13.2.8.4. Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word ALL, which must follow a comparison operator, means “return TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table t1 containing (10). The expression is TRUE if table t2 contains (-5, 0, +5) because 10 is greater than all three values in t2. The expression is FALSE if table t2 contains (12, 6, NULL, -100) because there is a single value 12 in table t2 that is greater than 10. The expression is *unknown* (that is, NULL) if table t2 contains (0, NULL, 1).

Finally, if table t2 is empty, the result is TRUE. So, the following statement is TRUE when table t2 is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this statement is NULL when table t2 is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following statement is NULL when table t2 is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables containing NULL values* and *empty tables* are “edge cases.” When writing subquery code, always consider whether you have taken those two possibilities into account.

NOT IN is an alias for `<> ALL`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);  
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.2.8.5. Row Subqueries

The discussion to this point has been of scalar or column subqueries; that is, subqueries that return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Here are two examples:

```
SELECT * FROM t1 WHERE (1,2) = (SELECT column1, column2 FROM t2);  
SELECT * FROM t1 WHERE ROW(1,2) = (SELECT column1, column2 FROM t2);
```

The queries here are both TRUE if table t2 has a row where column1 = 1 and column2 = 2.

The expressions (1, 2) and ROW(1, 2) are sometimes called *row constructors*. The two are equivalent. They are legal in other contexts as well. For example, the following two statements are semantically equivalent (although currently only the second one can be optimized):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);  
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

The normal use of row constructors is for comparisons with subqueries that return two or more columns. For example, the following query answers the request, “find all rows in table t1 that also exist in table t2”:

```
SELECT column1, column2, column3  
FROM t1  
WHERE (column1, column2, column3) IN  
(SELECT column1, column2, column3 FROM t2);
```

13.2.8.6. EXISTS and NOT EXISTS

If a subquery returns any rows at all, EXISTS subquery is TRUE, and NOT EXISTS subquery is FALSE. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an EXISTS subquery starts with SELECT *, but it could begin with SELECT 5 or SELECT column1 or anything at all. MySQL ignores the SELECT list in such a subquery, so it makes no difference.

For the preceding example, if t2 contains any rows, even rows with nothing but NULL values, the EXISTS condition is TRUE. This is actually an unlikely example because a [NOT] EXISTS subquery almost always contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE EXISTS (SELECT * FROM cities_stores
                WHERE cities_stores.store_type = stores.store_ty
```

- What kind of store is present in no cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE NOT EXISTS (SELECT * FROM cities_stores
                    WHERE cities_stores.store_type = stores.stor
```

- What kind of store is present in all cities?

```
SELECT DISTINCT store_type FROM stores s1
  WHERE NOT EXISTS (
    SELECT * FROM cities WHERE NOT EXISTS (
      SELECT * FROM cities_stores
        WHERE cities_stores.city = cities.city
        AND cities_stores.store_type = stores.store_type));
```

The last example is a double-nested NOT EXISTS query. That is, it has a NOT EXISTS clause within a NOT EXISTS clause. Formally, it answers the question “does a city exist with a store that is not in Stores”? But it is easier to say that a nested NOT EXISTS answers the question “is x TRUE for all y?”

13.2.8.7. Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1 WHERE column1 = ANY
(SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of t1, even though the subquery's FROM clause does not mention a table t1. So, MySQL looks outside the subquery, and finds t1 in the outer query.

Suppose that table `t1` contains a row where `column1 = 5` and `column2 = 6`; meanwhile, table `t2` contains a row where `column1 = 5` and `column2 = 7`. The simple expression `... WHERE column1 = ANY (SELECT column1 FROM t2)` would be `TRUE`, but in this example, the `WHERE` clause within the subquery is `FALSE` (because `(5, 6)` is not equal to `(5, 7)`), so the subquery as a whole is `FALSE`.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
WHERE x.column1 = (SELECT column1 FROM t2 AS x
WHERE x.column1 = (SELECT column1 FROM t3
WHERE x.column2 = t3.column1));
```

In this statement, `x.column2` must be a column in table `t2` because `SELECT column1 FROM t2 AS x ...` renames `t2`. It is not a column in table `t1` because `SELECT column1 FROM t1 ...` is an outer query that is *farther out*.

For subqueries in `HAVING` or `ORDER BY` clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Correlated subqueries cannot refer to the results of aggregate functions from the outer query.

13.2.8.8. Subqueries in the `FROM` clause

Subqueries are legal in a `SELECT` statement's `FROM` clause. The actual syntax is:

```
SELECT ... FROM (subquery) [AS] name ...
```

The `[AS] name` clause is mandatory, because every table in a `FROM` clause must have a name. Any columns in the `subquery` select list must have unique names. You can find this syntax described elsewhere in this manual, where the term used is “derived tables.”

For the sake of illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here is how to use a subquery in the FROM clause, using the example table:

```
INSERT INTO t1 VALUES (1, '1', 1.0);
INSERT INTO t1 VALUES (2, '2', 2.0);
SELECT sb1, sb2, sb3
FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
WHERE sb1 > 1;
```

Result: 2, '2', 4.0.

Here is another example: Suppose that you want to know the average of a set of sums for a grouped table. This does not work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

However, this query provides the desired information:

```
SELECT AVG(sum_column1)
FROM (SELECT SUM(column1) AS sum_column1
FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery (sum_column1) is recognized in the outer query.

Subqueries in the FROM clause can return a scalar, column, row, or table. Subqueries in the FROM clause cannot be correlated subqueries.

Subqueries in the FROM clause are executed even for the EXPLAIN statement (that is, derived temporary tables are built). This occurs because upper level queries need information about all tables during optimization phase.

13.2.8.9. Subquery Errors

There are some errors that apply only to subqueries. This section describes them.

- Unsupported subquery syntax:

```
ERROR 1235 (ER_NOT_SUPPORTED_YET)
SQLSTATE = 42000
```

```
Message = "This version of MySQL does not yet support  
'LIMIT & IN/ALL/ANY/SOME subquery'"
```

This means that statements of the following form do not work yet:

```
SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIM1
```

- Incorrect number of columns from subquery:

```
ERROR 1241 (ER_OPERAND_COL)  
SQLSTATE = 21000  
Message = "Operand should contain 1 column(s)"
```

This error occurs in cases like this:

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

You may use a subquery that returns multiple columns, if the purpose is comparison. See [Section 13.2.8.5, “Row Subqueries”](#). However, in other contexts, the subquery must be a scalar operand.

- Incorrect number of rows from subquery:

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)  
SQLSTATE = 21000  
Message = "Subquery returns more than 1 row"
```

This error occurs for statements where the subquery returns more than one row. Consider the following example:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

If `SELECT column1 FROM t2` returns just one row, the previous query will work. If the subquery returns more than one row, error 1242 will occur. In that case, the query should be rewritten as:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Incorrectly used table in subquery:

```
Error 1093 (ER_UPDATE_TABLE_USED)  
SQLSTATE = HY000  
Message = "You can't specify target table 'x'  
for update in FROM clause"
```

This error occurs in cases such as the following:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

You can use a subquery for assignment within an UPDATE statement because subqueries are legal in UPDATE and DELETE statements as well as in SELECT statements. However, you cannot use the same table (in this case, table t1) for both the subquery's FROM clause and the update target.

For transactional storage engines, the failure of a subquery causes the entire statement to fail. For non-transactional storage engines, data modifications made before the error was encountered are preserved.

13.2.8.10. Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. The following list provides some interesting tricks that you might want to play with:

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

```
SELECT * FROM t1 WHERE t1.column1 IN
(SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
(SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
(SELECT * FROM t2 LIMIT 1);
```

- Replace a join with a subquery. For example, try this:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
SELECT column1 FROM t2);
```

Instead of this:

```
SELECT DISTINCT t1.column1 FROM t1, t2
WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL that do not support subqueries. However, in some cases, converting a subquery to a join may improve performance. See [Section 13.2.8.11, “Rewriting Subqueries as Joins for Earlier MySQL](#)

Versions”.

- Move clauses from outside to inside the subquery. For example, use this query:

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

For another example, use this query:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

Instead of this query:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use a row subquery instead of a correlated subquery. For example, use this query:

```
SELECT * FROM t1
WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
AND t2.column2=t1.column2);
```

- Use NOT (a = ANY (...)) rather than a <> ALL (...).
- Use x = ANY (table containing (1,2)) rather than x=1 OR x=2.
- Use = ANY rather than EXISTS.
- For uncorrelated subqueries that always return one row, IN is always slower than =. For example, use this query:

```
SELECT * FROM t1 WHERE t1.col_name
= (SELECT a FROM t2 WHERE b = some_const);
```

Instead of this query:

```
SELECT * FROM t1 WHERE t1.col_name
IN (SELECT a FROM t2 WHERE b = some_const);
```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the `BENCHMARK()` function, you can get an idea about what helps in your own situation. See [Section 12.9.3, “Information Functions”](#).

Some optimizations that MySQL itself makes are:

- MySQL executes non-correlated subqueries only once. Use `EXPLAIN` to make sure that a given subquery really is non-correlated.
- MySQL rewrites `IN`, `ALL`, `ANY`, and `SOME` subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which `EXPLAIN` describes as a special join type (`unique_subquery` or `index_subquery`):

```
... IN (SELECT indexed_column FROM single_table ...)
```

- MySQL enhances expressions of the following form with an expression involving `MIN()` or `MAX()`, unless `NULL` values or empty sets are involved:

```
value {ALL|ANY|SOME} {> | < | >= | <=} (non-correlated subquery)
```

For example, this `WHERE` clause:

```
WHERE 5 > ALL (SELECT x FROM t)
```

might be treated by the optimizer like this:

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

There is a chapter titled “How MySQL Transforms Subqueries” in the MySQL Internals Manual, available at <http://dev.mysql.com/doc/>.

13.2.8.11. Rewriting Subqueries as Joins for Earlier MySQL Versions

In previous versions of MySQL (prior to MySQL 4.1), only nested queries of the form `INSERT ... SELECT ...` and `REPLACE ... SELECT ...` were supported. Although this is not the case in MySQL 5.0, it is still true that there are sometimes other ways to test membership in a set of values. It is also true that on some occasions, it is not only possible to rewrite a query without a subquery, but it can be more efficient to make use of some of these techniques rather than to use subqueries. One of these is the `IN()` construct:

For example, this query:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);  
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.
```

Can be rewritten using `IN()`:

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id  
WHERE table2.id IS NULL;
```

A `LEFT [OUTER] JOIN` can be faster than an equivalent subquery because the server might be able to optimize it better — a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not exist, so subqueries were the only way to do certain things. Today, MySQL Server and many other modern database systems offer a wide range of outer join types.

MySQL Server supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported.

13.2.9. TRUNCATE Syntax

```
TRUNCATE [TABLE] tbl_name
```

`TRUNCATE TABLE` empties a table completely. Logically, this is equivalent to a

DELETE statement that deletes all rows, but there are practical differences under some circumstances.

For InnoDB before version 5.0.3, TRUNCATE TABLE is mapped to DELETE, so there is no difference. Starting with MySQL 5.0.3, fast TRUNCATE TABLE is available. However, the operation is still mapped to DELETE if there are foreign key constraints that reference the table. (When fast truncate is used, it resets any AUTO_INCREMENT counter. From MySQL 5.0.13 on, the AUTO_INCREMENT counter is reset by TRUNCATE TABLE, regardless of whether there is a foreign key constraint.)

For other storage engines, TRUNCATE TABLE differs from DELETE in the following ways in MySQL 5.0:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one.
- Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock.
- The number of deleted rows is not returned.
- As long as the table format file `tbl_name.frm` is valid, the table can be re-created as an empty table with TRUNCATE TABLE, even if the data or index files have become corrupted.
- The table handler does not remember the last used AUTO_INCREMENT value, but starts counting from the beginning. This is true even for MyISAM and InnoDB, which normally do not reuse sequence values.

Since truncation of a table does not make any use of DELETE, the TRUNCATE statement does not invoke ON DELETE triggers.

TRUNCATE TABLE is an Oracle SQL extension adopted in MySQL.

13.2.10. UPDATE Syntax

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
```

```
SET col_name1=expr1 [, col_name2=expr2 ...]  
[WHERE where_condition]  
[ORDER BY ...]  
[LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references  
SET col_name1=expr1 [, col_name2=expr2 ...]  
[WHERE where_condition]
```

For the single-table syntax, the UPDATE statement updates columns of existing rows in *tbl_name* with new values. The SET clause indicates which columns to modify and the values they should be given. The WHERE clause, if given, specifies the conditions that identify which rows to update. With no WHERE clause, all rows are updated. If the ORDER BY clause is specified, the rows are updated in the order that is specified. The LIMIT clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, UPDATE updates rows in each table named in *table_references* that satisfy the conditions. In this case, ORDER BY and LIMIT cannot be used.

where_condition is an expression that evaluates to true for each row to be updated. It is specified as described in [Section 13.2.7, “SELECT Syntax”](#).

The UPDATE statement supports the following modifiers:

- If you use the LOW_PRIORITY keyword, execution of the UPDATE is delayed until no other clients are reading from the table.
- If you use the IGNORE keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur are not updated. Rows for which columns are updated to values that would cause data conversion errors are updated to the closest valid values instead.

If you access a column from *tbl_name* in an expression, UPDATE uses the current value of the column. For example, the following statement sets the age column to one more than its current value:

```
UPDATE persondata SET age=age+1;
```

Single-table UPDATE assignments are generally evaluated from left to right. For multiple-table updates, there is no guarantee that assignments are carried out in any particular order.

If you set a column to the value it currently has, MySQL notices this and does not update it.

If you update a column that has been declared NOT NULL by setting to NULL, the column is set to the default value appropriate for the data type and the warning count is incremented. The default value is 0 for numeric types, the empty string (' ') for string types, and the “zero” value for date and time types.

UPDATE returns the number of rows that were actually changed. The `mysql_info()` C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the UPDATE.

You can use `LIMIT row_count` to restrict the scope of the UPDATE. A `LIMIT` clause is a rows-matched restriction. The statement stops as soon as it has found `row_count` rows that satisfy the `WHERE` clause, whether or not they actually were changed.

If an UPDATE statement includes an `ORDER BY` clause, the rows are updated in the order specified by the clause. This can be useful in certain situations that might otherwise result in an error. Suppose that a table `t` contains a column `id` that has a unique index. The following statement could fail with a duplicate-key error, depending on the order in which rows are updated:

```
UPDATE t SET id = id + 1;
```

For example, if the table contains 1 and 2 in the `id` column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an `ORDER BY` clause to cause the rows with larger `id` values to be updated before those with smaller values:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

You can also perform UPDATE operations covering multiple tables. However, you cannot use `ORDER BY` or `LIMIT` with a multiple-table UPDATE. The `table_references` clause lists the tables involved in the join. Its syntax is described in [Section 13.2.7.1, “JOIN Syntax”](#). Here is an example:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The preceding example shows an inner join that uses the comma operator, but multiple-table UPDATE statements can use any type of join allowed in SELECT statements, such as LEFT JOIN.

You need the UPDATE privilege only for columns referenced in a multiple-table UPDATE that are actually updated. You need only the SELECT privilege for any columns that are read but not modified.

If you use a multiple-table UPDATE statement involving InnoDB tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the ON UPDATE capabilities that InnoDB provides to cause the other tables to be modified accordingly. See [Section 14.2.6.4, “FOREIGN KEY Constraints”](#).

Currently, you cannot update a table and select from the same table in a subquery.

13.3. MySQL Utility Statements

13.3.1. DESCRIBE Syntax

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

DESCRIBE provides information about the columns in a table. It is a shortcut for SHOW COLUMNS FROM. As of MySQL 5.0.1, these statements also display information for views. (See [Section 13.5.4.3, “SHOW COLUMNS Syntax”](#).)

col_name can be a column name, or a string containing the SQL ‘%’ and ‘_’ wildcard characters to obtain output only for the columns with names matching the string. There is no need to enclose the string within quotes unless it contains spaces or other special characters.

```
mysql> DESCRIBE city;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null  | Key  | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| Id         | int(11)   | NO    | PRI  | NULL    | auto_increment |
| Name       | char(35)  | NO    |      |          |                 |
| Country    | char(3)   | NO    | UNI  |          |                 |
| District   | char(20)  | YES   | MUL  |          |                 |
| Population | int(11)   | NO    |      | 0       |                 |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Field indicates the column name.

The Null field indicates whether NULL values can be stored in the column.

The key field indicates whether the column is indexed. A value of PRI indicates that the column is part of the table's primary key. UNI indicates that the column is part of a UNIQUE index. The MUL value indicates that multiple occurrences of a given value are allowed within the column.

One reason for MUL to be displayed on a UNIQUE index is that several columns form a composite UNIQUE index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value. Note that in a composite index, only the leftmost column of the index has an entry in the key field.

Before MySQL 5.0.11, if the column allows NULL values, the key value can be MUL even when a UNIQUE index is used. The rationale was that multiple rows in a UNIQUE index can hold a NULL value if the column is not declared NOT NULL. As of MySQL 5.0.11, the display is UNI rather than MUL regardless of whether the column allows NULL; you can see from the Null field whether or not the column can contain NULL.

The Default field indicates the default value that is assigned to the column.

The Extra field contains any additional information that is available about a given column. In the example shown, the Extra field indicates that the Id column was created with the AUTO_INCREMENT keyword.

If the data types are different from what you expect them to be based on a CREATE TABLE statement, note that MySQL sometimes changes data types. See [Section 13.1.5.1, “Silent Column Specification Changes”](#).

The DESCRIBE statement is provided for compatibility with Oracle.

The SHOW CREATE TABLE and SHOW TABLE STATUS statements also provide information about tables. See [Section 13.5.4, “SHOW Syntax”](#).

13.3.2. HELP Syntax

```
HELP 'search_string'
```

The HELP statement returns online information from the MySQL Reference manual. Its proper operation requires that the help tables in the mysql database be initialized with help topic information (see [Section 5.2.7, “MySQL Server-Side Help Support”](#)).

The HELP statement searches the help tables for the given search string and displays the result of the search. The search string is not case sensitive.

The HELP statement understands several types of search strings:

- At the most general level, use contents to retrieve a list of the top-level help categories:

```
HELP 'contents'
```

- For a list of topics in a given help category, such as Data Types, use the category name:

```
HELP 'data types'
```

- For help on a specific help topic, such as the ASCII() function or the CREATE TABLE statement, use the associated keyword or keywords:

```
HELP 'ascii'  
HELP 'create table'
```

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string will return a list of items or the help information for a single help topic. However, you can tell what kind of response HELP returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the **mysql** client, but note that **mysql** itself reformats HELP result sets in a different way.

- Empty result set

No match could be found for the search string.

- Result set containing a single row with three columns

This means that the search string yielded a hit for the help topic. The result has three columns:

- name: The topic name.
- description: Descriptive help text for the topic.
- example: Usage example or examples. This column might be blank.

Example: `HELP 'replace'`

Yields:

name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_s replaced by the string to_str. REPLACE() performs a case-sensitive match when searching for from_str.

example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'

- Result set containing multiple rows with two columns

This means that the search string matched many help topics. The result set indicates the help topic names:

- name: The help topic name.
- is_it_category: Y if the name represents a help category, N if it does not. If it does not, the name value when specified as the argument to the HELP statement should yield a single-row result set containing a description for the named item.

Example: HELP 'status'

Yields:

name	is_it_category
SHOW	N
SHOW ENGINE	N
SHOW INNODB STATUS	N
SHOW MASTER STATUS	N
SHOW PROCEDURE STATUS	N
SHOW SLAVE STATUS	N
SHOW STATUS	N
SHOW TABLE STATUS	N

- Result set containing multiple rows with three columns

This means the search string matches a category. The result set contains category entries:

- source_category_name: The help category name.

- name: The category or topic name
- is_it_category: Y if the name represents a help category, N if it does not. If it does not, the name value when specified as the argument to the HELP statement should yield a single-row result set containing a description for the named item.

Example: HELP 'functions'

Yields:

source_category_name	name	is_it_category
Functions	CREATE FUNCTION	N
Functions	DROP FUNCTION	N
Functions	Bit Functions	Y
Functions	Comparison operators	Y
Functions	Control flow functions	Y
Functions	Date and Time Functions	Y
Functions	Encryption Functions	Y
Functions	Information Functions	Y
Functions	Logical operators	Y
Functions	Miscellaneous Functions	Y
Functions	Numeric Functions	Y
Functions	String Functions	Y

13.3.3. USE Syntax

USE *db_name*

The USE *db_name* statement tells MySQL to use the *db_name* database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another USE statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;    # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;    # selects from db2.mytable
```

Making a particular database the default by means of the USE statement does not preclude you from accessing tables in other databases. The following example

accesses the author table from the db1 database and the editor table from the db2 database:

```
USE db1;  
SELECT author_name,editor_name FROM author,db2.editor  
WHERE author.editor_id = db2.editor.editor_id;
```

The USE statement is provided for compatibility with Sybase.

13.4. MySQL Transactional and Locking Statements

MySQL supports local transactions (within a given client connection) through statements such as `SET AUTOCOMMIT`, `START TRANSACTION`, `COMMIT`, and `ROLLBACK`. See [Section 13.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#). Beginning with MySQL 5.0, XA transaction support is available, which enables MySQL to participate in distributed transactions as well. See [Section 13.4.7, “XA Transactions”](#).

13.4.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax

```
START TRANSACTION | BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET AUTOCOMMIT = {0 | 1}
```

The `START TRANSACTION` and `BEGIN` statement begin a new transaction. `COMMIT` commits the current transaction, making its changes permanent. `ROLLBACK` rolls back the current transaction, canceling its changes. The `SET AUTOCOMMIT` statement disables or enables the default autocommit mode for the current connection.

Beginning with MySQL 5.0.3, the optional `WORK` keyword is supported for `COMMIT` and `RELEASE`, as are the `CHAIN` and `RELEASE` clauses. `CHAIN` and `RELEASE` can be used for additional control over transaction completion. The value of the `completion_type` system variable determines the default completion behavior. See [Section 5.2.2, “Server System Variables”](#).

The `AND CHAIN` clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The `RELEASE` clause causes the server to disconnect the current client connection after terminating the current transaction. Including the `NO` keyword suppresses `CHAIN` or `RELEASE` completion, which can be useful if the `completion_type` system variable is set to cause chaining or release completion by default.

By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores

the update on disk.

If you are using a transaction-safe storage engine (such as InnoDB, BDB, or NDB Cluster), you can disable autocommit mode with the following statement:

```
SET AUTOCOMMIT=0;
```

After disabling autocommit mode by setting the AUTOCOMMIT variable to zero, you must use COMMIT to store your changes to disk or ROLLBACK if you want to ignore the changes you have made since the beginning of your transaction.

To disable autocommit mode for a single series of statements, use the START TRANSACTION statement:

```
START TRANSACTION;  
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;  
UPDATE table2 SET summary=@A WHERE type=1;  
COMMIT;
```

With START TRANSACTION, autocommit remains disabled until you end the transaction with COMMIT or ROLLBACK. The autocommit mode then reverts to its previous state.

BEGIN and BEGIN WORK are supported as aliases of START TRANSACTION for initiating a transaction. START TRANSACTION is standard SQL syntax and is the recommended way to start an ad-hoc transaction.

The BEGIN statement differs from the use of the BEGIN keyword that starts a BEGIN . . . END compound statement. The latter does not begin a transaction. See [Section 17.2.5, “BEGIN . . . END Compound Statement Syntax”](#).

You can also begin a transaction like this:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

The WITH CONSISTENT SNAPSHOT clause starts a consistent read for storage engines that are capable of it. Currently, this applies only to InnoDB. The effect is the same as issuing a START TRANSACTION followed by a SELECT from any InnoDB table. See [Section 14.2.10.4, “Consistent Non-Locking Read”](#).

The WITH CONSISTENT SNAPSHOT clause does not change the current transaction isolation level, so it provides a consistent snapshot only if the current isolation

level is one that allows consistent read (`REPEATABLE READ` or `SERIALIZABLE`).

Beginning a transaction causes an implicit `UNLOCK TABLES` to be performed.

For best results, transactions should be performed using only tables managed by a single transactional storage engine. Otherwise, the following problems can occur:

- If you use tables from more than one transaction-safe storage engine (such as InnoDB and BDB), and the transaction isolation level is not `SERIALIZABLE`, it is possible that when one transaction commits, another ongoing transaction that uses the same tables will see only some of the changes made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use `SET TRANSACTION ISOLATION LEVEL` to set the isolation level to `SERIALIZABLE` on a per-transaction basis as necessary.)
- If you use non-transaction-safe tables within a transaction, any changes to those tables are stored at once, regardless of the status of autocommit mode.

If you issue a `ROLLBACK` statement after updating a non-transactional table within a transaction, an `ER_WARNING_NOT_COMPLETE_ROLLBACK` warning occurs. Changes to transaction-safe tables are rolled back, but not changes to non-transaction-safe tables.

Each transaction is stored in the binary log in one chunk, upon `COMMIT`. Transactions that are rolled back are not logged. (**Exception:** Modifications to non-transactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to non-transactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that the modifications to those tables are replicated.) See [Section 5.12.3, “The Binary Log”](#).

You can change the isolation level for transactions with `SET TRANSACTION ISOLATION LEVEL`. See [Section 13.4.6, “SET TRANSACTION Syntax”](#).

Rolling back can be a slow operation that may occur without the user having explicitly asked for it (for example, when an error occurs). Because of this, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the connection during implicit and explicit (`ROLLBACK SQL` statement) rollbacks.

13.4.2. Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a ROLLBACK statement.

13.4.3. Statements That Cause an Implicit Commit

Each of the following statements (and any synonyms for them) implicitly end a transaction, as if you had done a COMMIT before executing the statement:

- ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, BEGIN, CREATE DATABASE, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, DROP DATABASE, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, LOAD MASTER DATA, LOCK TABLES, LOAD DATA INFILE, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE TABLE, UNLOCK TABLES.
- UNLOCK TABLES commits a transaction only if any tables currently are locked.
- The CREATE TABLE, CREATE DATABASE DROP DATABASE, and TRUNCATE TABLE statements cause an implicit commit beginning with MySQL 5.0.8. The ALTER FUNCTION, ALTER PROCEDURE, CREATE FUNCTION, CREATE PROCEDURE, DROP FUNCTION, and DROP PROCEDURE statements cause an implicit commit beginning with MySQL 5.0.13.
- The CREATE TABLE statement in InnoDB is processed as a single transaction. This means that a ROLLBACK from the user does not undo CREATE TABLE statements the user made during that transaction.

Transactions cannot be nested. This is a consequence of the implicit COMMIT performed for any current transaction when you issue a START TRANSACTION statement or one of its synonyms.

Statements that cause implicit cannot be used in an XA transaction while the transaction is in an ACTIVE state.

13.4.4. SAVEPOINT and ROLLBACK TO SAVEPOINT Syntax

```
SAVEPOINT identifier  
ROLLBACK [WORK] TO SAVEPOINT identifier  
RELEASE SAVEPOINT identifier
```

InnoDB supports the SQL statements SAVEPOINT and ROLLBACK TO SAVEPOINT. Starting from MySQL 5.0.3, RELEASE SAVEPOINT and the optional WORK keyword for ROLLBACK are supported as well.

The SAVEPOINT statement sets a named transaction savepoint with a name of *identifier*. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The ROLLBACK TO SAVEPOINT statement rolls back a transaction to the named savepoint. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but InnoDB does *not* release the row locks that were stored in memory after the savepoint. (Note that for a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the ROLLBACK TO SAVEPOINT statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1181: Got error 153 during ROLLBACK
```

The RELEASE SAVEPOINT statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.

All savepoints of the current transaction are deleted if you execute a COMMIT, or a ROLLBACK that does not name a savepoint.

Beginning with MySQL 5.0.17, a new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels

become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

13.4.5. LOCK TABLES and UNLOCK TABLES Syntax

LOCK TABLES

```
tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
```

```
[, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ..
```

UNLOCK TABLES

LOCK TABLES locks base tables (but not views) for the current thread. If any of the tables are locked by other threads, it blocks until all locks can be acquired. UNLOCK TABLES releases any locks held by the current thread. All tables that are locked by the current thread are implicitly unlocked when the thread issues another LOCK TABLES, or when the connection to the server is closed.

A table lock protects only against inappropriate reads or writes by other clients. The client holding the lock, even a read lock, can perform table-level operations such as DROP TABLE.

Note the following regarding the use of LOCK TABLES with transactional tables:

- LOCK TABLES is not transaction-safe and implicitly commits any active transactions before attempting to lock the tables. Also, beginning a transaction (for example, with START TRANSACTION) implicitly performs an UNLOCK TABLES. (See [Section 13.4.3, “Statements That Cause an Implicit Commit”](#).)
- The correct way to use LOCK TABLES with transactional tables, such as InnoDB tables, is to set AUTOCOMMIT = 0 and not to call UNLOCK TABLES until you commit the transaction explicitly. When you call LOCK TABLES, InnoDB internally takes its own table lock, and MySQL takes its own table lock. InnoDB releases its table lock at the next commit, but for MySQL to release its table lock, you have to call UNLOCK TABLES. You should not have AUTOCOMMIT = 1, because then InnoDB releases its table lock immediately after the call of LOCK TABLES, and deadlocks can very easily happen. Note that we do not acquire the InnoDB table lock at all if AUTOCOMMIT=1, to help old applications avoid unnecessary deadlocks.

- ROLLBACK does not release MySQL's non-transactional table locks.

To use `LOCK TABLES`, you must have the `LOCK TABLES` privilege and the `SELECT` privilege for the involved tables.

The main reasons to use `LOCK TABLES` are to emulate transactions or to get more speed when updating tables. This is explained in more detail later.

If a thread obtains a `READ` lock on a table, that thread (and all other threads) can only read from the table. If a thread obtains a `WRITE` lock on a table, only the thread holding the lock can write to the table. Other threads are blocked from reading or writing the table until the lock has been released.

The difference between `READ LOCAL` and `READ` is that `READ LOCAL` allows non-conflicting `INSERT` statements (concurrent inserts) to execute while the lock is held. However, this cannot be used if you are going to manipulate the database files outside MySQL while you hold the lock. For InnoDB tables, `READ LOCAL` is the same as `READ` as of MySQL 5.0.13. (Before that, `READ LOCAL` essentially does nothing: It does not lock the table at all, so for InnoDB tables, the use of `READ LOCAL` is deprecated because a plain consistent-read `SELECT` does the same thing, and no locks are needed.)

When you use `LOCK TABLES`, you must lock all tables that you are going to use in your queries. Because `LOCK TABLES` will not lock views, if the operation that you are performing uses any views, you must also lock all of the base tables on which those views depend. While the locks obtained with a `LOCK TABLES` statement are in effect, you cannot access any tables that were not locked by the statement. Also, you cannot use a locked table multiple times in a single query. Use aliases instead, in which case you must obtain a lock for each alias separately.

```
mysql> LOCK TABLE t WRITE, t AS t1 WRITE;  
mysql> INSERT INTO t SELECT * FROM t;  
ERROR 1100: Table 't' was not locked with LOCK TABLES  
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

If your queries refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
```

```
mysql> SELECT * FROM t AS myalias;  
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your queries using that alias:

```
mysql> LOCK TABLE t AS myalias READ;  
mysql> SELECT * FROM t;  
ERROR 1100: Table 't' was not locked with LOCK TABLES  
mysql> SELECT * FROM t AS myalias;
```

WRITE locks normally have higher priority than READ locks to ensure that updates are processed as soon as possible. This means that if one thread obtains a READ lock and then another thread requests a WRITE lock, subsequent READ lock requests wait until the WRITE thread has gotten the lock and released it. You can use LOW_PRIORITY WRITE locks to allow other threads to obtain READ locks while the thread is waiting for the WRITE lock. You should use LOW_PRIORITY WRITE locks only if you are sure that eventually there will be a time when no threads have a READ lock.

LOCK TABLES works as follows:

1. Sort all tables to be locked in an internally defined order. From the user standpoint, this order is undefined.
2. If a table is locked with a read and a write lock, put the write lock before the read lock.
3. Lock one table at a time until the thread gets all locks.

This policy ensures that table locking is deadlock free. There are, however, other things you need to be aware of about this policy: If you are using a LOW_PRIORITY WRITE lock for a table, it means only that MySQL waits for this particular lock until there are no threads that want a READ lock. When the thread has gotten the WRITE lock and is waiting to get the lock for the next table in the lock table list, all other threads wait for the WRITE lock to be released. If this becomes a serious problem with your application, you should consider converting some of your tables to transaction-safe tables.

You can safely use KILL to terminate a thread that is waiting for a table lock. See [Section 13.5.5.3, “KILL Syntax”](#).

Note that you should *not* lock any tables that you are using with INSERT DELAYED because in that case the INSERT is performed by a separate thread.

Normally, you do not need to lock tables, because all single UPDATE statements are atomic; no other thread can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

- If you are going to run many operations on a set of MyISAM tables, it is much faster to lock the tables you are going to use. Locking MyISAM tables speeds up inserting, updating, or deleting on them. The downside is that no thread can update a READ-locked table (including the one holding the lock) and no thread can access a WRITE-locked table other than the one holding the lock.

The reason some MyISAM operations are faster under LOCK TABLES is that MySQL does not flush the key cache for the locked tables until UNLOCK TABLES is called. Normally, the key cache is flushed after each SQL statement.

- If you are using a storage engine in MySQL that does not support transactions, you must use LOCK TABLES if you want to ensure that no other thread comes between a SELECT and an UPDATE. The example shown here requires LOCK TABLES to execute safely:

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

Without LOCK TABLES, it is possible that another thread might insert a new row in the trans table between execution of the SELECT and UPDATE statements.

You can avoid using LOCK TABLES in many cases by using relative updates (UPDATE customer SET value=value+new_value) or the LAST_INSERT_ID() function. See [Section 1.9.5.3, “Transactions and Atomic Operations”](#).

You can also avoid locking tables in some cases by using the user-level advisory lock functions GET_LOCK() and RELEASE_LOCK(). These locks are saved in a hash

table in the server and implemented with `pthread_mutex_lock()` and `pthread_mutex_unlock()` for high speed. See [Section 12.9.4, “Miscellaneous Functions”](#).

See [Section 7.3.1, “Locking Methods”](#), for more information on locking policy.

You can lock all tables in all databases with read locks with the `FLUSH TABLES WITH READ LOCK` statement. See [Section 13.5.5.2, “FLUSH Syntax”](#). This is a very convenient way to get backups if you have a filesystem such as Veritas that can take snapshots in time.

Note: If you use `ALTER TABLE` on a locked table, it may become unlocked. See [Section A.7.1, “Problems with ALTER TABLE”](#).

13.4.6. SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL  
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE
```

This statement sets the transaction isolation level for the next transaction, globally, or for the current session.

The default behavior of `SET TRANSACTION` is to set the isolation level for the next (not yet started) transaction. If you use the `GLOBAL` keyword, the statement sets the default transaction level globally for all new connections created from that point on. Existing connections are unaffected. You need the `SUPER` privilege to do this. Using the `SESSION` keyword sets the default transaction level for all future transactions performed on the current connection.

For descriptions of each InnoDB transaction isolation level, see [Section 14.2.10.3, “InnoDB and TRANSACTION ISOLATION LEVEL”](#). InnoDB supports each of these levels in MySQL 5.0. The default level is `REPEATABLE READ`.

To set the initial default global isolation level for `mysqld`, use the `--transaction-isolation` option. See [Section 5.2.1, “mysqld Command Options”](#).

13.4.7. XA Transactions

MySQL 5.0.3 and up provides server-side support for XA transactions. Currently, this support is available for the InnoDB storage engine. The MySQL XA implementation is based on the X/Open CAE document *Distributed Transaction Processing: The XA Specification*. This document is published by The Open Group and available at <http://www.opengroup.org/public/pubs/catalog/c193.htm>. Limitations of the current XA implementation are described in [Section I.5, “Restrictions on XA Transactions”](#).

On the client side, there are no special requirements. The XA interface to a MySQL server consists of SQL statements that begin with the XA keyword. MySQL client programs must be able to send SQL statements and to understand the semantics of the XA statement interface. They do not need be linked against a recent client library. Older client libraries also will work.

Currently, among the MySQL Connectors, MySQL Connector/J 5.0.0 supports XA directly (by means of a class interface that handles the XA SQL statement interface for you).

XA supports distributed transactions; that is, the ability to allow multiple separate transactional resources to participate in a global transaction. Transactional resources often are RDBMSs but may be other kinds of resources.

A global transaction involves several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. (However, for a distributed transaction, you must use the SERIALIZABLE isolation level to achieve ACID properties. It is enough to use REPEATABLE READ for a non-distributed transaction, but not for a distributed transaction.)

Some examples of distributed transactions:

- An application may act as an integration tool that combines a messaging service with an RDBMS. The application makes sure that transactions dealing with message sending, retrieval, and processing that also involve a transactional database all happen in a global transaction. You can think of this as “transactional email.”

- An application performs actions that involve different database servers, such as a MySQL server and an Oracle server (or multiple MySQL servers), where actions that involve multiple servers must happen as part of a global transaction, rather than as separate transactions local to each server.
- A bank keeps account information in an RDBMS and distributes and receives money via automated teller machines (ATMs). It is necessary to ensure that ATM actions are correctly reflected in the accounts, but this cannot be done with the RDBMS alone. A global transaction manager integrates the ATM and database resources to ensure overall consistency of financial transactions.

Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

- A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.
- A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are “branches” of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

The MySQL implementation of XA MySQL enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.

To carry out a global transaction, it is necessary to know which components are involved, and bring each component to a point when it can be committed or rolled back. Depending on what each component reports about its ability to succeed, they must all commit or roll back as an atomic group. That is, either all components must commit, or all components must roll back. To manage a global transaction, it is necessary to take into account that any component or the connecting network might fail.

The process for executing a global transaction uses two-phase commit (2PC). This takes place after the actions performed by the branches of the global

transaction have been executed.

1. In the first phase, all branches are prepared. That is, they are told by the TM to get ready to commit. Typically, this means each RM that manages a branch records the actions for the branch in stable storage. The branches indicate whether they are able to do this, and these results are used for the second phase.
2. In the second phase, the TM tells the RMs whether to commit or roll back. If all branches indicated when they were prepared that they will be able to commit, all branches are told to commit. If any branch indicated when it was prepared that it will not be able to commit, all branches are told to roll back.

In some cases, a global transaction might use one-phase commit (1PC). For example, when a Transaction Manager finds that a global transaction consists of only one transactional resource (that is, a single branch), that resource can be told to prepare and commit at the same time.

13.4.7.1. XA Transaction SQL Syntax

To perform XA transactions in MySQL, use the following statements:

```
XA {START|BEGIN} xid [JOIN|RESUME]
```

```
XA END xid [SUSPEND [FOR MIGRATE]]
```

```
XA PREPARE xid
```

```
XA COMMIT xid [ONE PHASE]
```

```
XA ROLLBACK xid
```

```
XA RECOVER
```

For XA START, the JOIN and RESUME clauses are not supported.

For XA END the SUSPEND [FOR MIGRATE] clause is not supported.

Each XA statement begins with the XA keyword, and most of them require an *xid* value. An *xid* is an XA transaction identifier. It indicates which transaction the statement applies to. *xid* values are supplied by the client, or generated by the

MySQL server. An *xid* value has from one to three parts:

```
xid: gtrid [, bqual [, formatID ]]
```

gtrid is a global transaction identifier, *bqual* is a branch qualifier, and *formatID* is a number that identifies the format used by the *gtrid* and *bqual* values. As indicated by the syntax, *bqual* and *formatID* are optional. The default *bqual* value is '' if not given. The default *formatID* value is 1 if not given.

gtrid and *bqual* must be string literals, each up to 64 bytes (not characters) long. *gtrid* and *bqual* can be specified in several ways. You can use a quoted string ('ab'), hex string (@x6162, x'ab'), or bit value (b'nnnn').

formatID is an unsigned integer.

The *gtrid* and *bqual* values are interpreted in bytes by the MySQL server's underlying XA support routines. However, while an SQL statement containing an XA statement is being parsed, the server works with some specific character set. To be safe, write *gtrid* and *bqual* as hex strings.

xid values typically are generated by the Transaction Manager. Values generated by one TM must be different from values generated by other TMs. A given TM must be able to recognize its own *xid* values in a list of values returned by the XA RECOVER statement.

XA START *xid* starts an XA transaction with the given *xid* value. Each XA transaction must have a unique *xid* value, so the value must not currently be used by another XA transaction. Uniqueness is assessed using the *gtrid* and *bqual* values. All following XA statements for the XA transaction must be specified using the same *xid* value as that given in the XA START statement. If you use any of those statements but specify an *xid* value that does not correspond to some existing XA transaction, an error occurs.

One or more XA transactions can be part of the same global transaction. All XA transactions within a given global transaction must use the same *gtrid* value in the *xid* value. For this reason, *gtrid* values must be globally unique so that there is no ambiguity about which global transaction a given XA transaction is part of. The *bqual* part of the *xid* value must be different for each XA transaction within a global transaction. (The requirement that *bqual* values be different is a limitation of the current MySQL XA implementation. It is not part of the XA

specification.)

The `XA RECOVER` statement returns information for those XA transactions on the MySQL server that are in the `PREPARED` state. (See [Section 13.4.7.2, “XA Transaction States”](#).) The output includes a row for each such XA transaction on the server, regardless of which client started it.

`XA RECOVER` output rows look like this (for an example `xid` value consisting of the parts 'abc', 'def', and 7):

```
mysql> XA RECOVER;
+-----+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data   |
+-----+-----+-----+-----+
|          7 |              3 |              3 | abcdef |
+-----+-----+-----+-----+
```

The output columns have the following meanings:

- `formatID` is the *formatID* part of the transaction *xid*
- `gtrid_length` is the length in bytes of the *gtrid* part of the *xid*
- `bqual_length` is the length in bytes of the *bqual* part of the *xid*
- `data` is the concatenation of the *gtrid* and *bqual* parts of the *xid*

13.4.7.2. XA Transaction States

An XA transaction progresses through the following states:

1. Use `XA START` to start an XA transaction and put it in the `ACTIVE` state.
2. For an `ACTIVE` XA transaction, issue the SQL statements that make up the transaction, and then issue an `XA END` statement. `XA END` puts the transaction in the `IDLE` state.
3. For an `IDLE` XA transaction, you can issue either an `XA PREPARE` statement or an `XA COMMIT ... ONE PHASE` statement:
 - `XA PREPARE` puts the transaction in the `PREPARED` state. An `XA RECOVER`

statement at this point will include the transaction's *xid* value in its output, because `XA RECOVER` lists all XA transactions that are in the PREPARED state.

- `XA COMMIT ... ONE PHASE` prepares and commits the transaction. The *xid* value will not be listed by `XA RECOVER` because the transaction terminates.

4. For a PREPARED XA transaction, you can issue an `XA COMMIT` statement to commit and terminate the transaction, or `XA ROLLBACK` to roll back and terminate the transaction.

Here is a simple XA transaction that inserts a row into a table as part of a global transaction:

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

Within the context of a given client connection, XA transactions and local (non-XA) transactions are mutually exclusive. For example, if `XA START` has been issued to begin an XA transaction, a local transaction cannot be started until the XA transaction has been committed or rolled back. Conversely, if a local transaction has been started with `START TRANSACTION`, no XA statements can be used until the transaction has been committed or rolled back.

Note that if an XA transaction is in the ACTIVE state, you cannot issue any statements that cause an implicit commit. That would violate the XA contract because you could not roll back the XA transaction. You will receive the following error if you try to execute such a statement:

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
```

when global transaction is in the ACTIVE state

Statements to which the preceding remark applies are listed at [Section 13.4.3, “Statements That Cause an Implicit Commit”](#).

13.5. Database Administration Statements

13.5.1. Account Management Statements

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in [Chapter 5, Database Administration](#), which you should consult for additional details.

Important: Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

13.5.1.1. CREATE USER Syntax

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']  
    [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

The `CREATE USER` statement was added in MySQL 5.0.2. This statement creates new MySQL accounts. To use it, you must have the global `CREATE USER` privilege or the `INSERT` privilege for the `mysql` database. For each account, `CREATE USER` creates a new record in the `mysql.user` table that has no privileges. An error occurs if the account already exists. Each account is named using the same format as for the `GRANT` statement; for example, `'jeffrey'@'localhost'`. The user and host parts of the account name correspond to the `User` and `Host` column values of the `user` table row for the account.

The account can be given a password with the optional `IDENTIFIED BY` clause. The `user` value and the password are given the same way as for the `GRANT` statement. In particular, to specify the password in plain text, omit the `PASSWORD` keyword. To specify the password as the hashed value as returned by the `PASSWORD()` function, include the `PASSWORD` keyword. See [Section 13.5.1.3, “GRANT Syntax”](#).

13.5.1.2. DROP USER Syntax

```
DROP USER user [, user] ...
```

The `DROP USER` statement removes one or more MySQL accounts. To use it, you must have the global `CREATE USER` privilege or the `DELETE` privilege for the `mysql` database. Each account is named using the same format as for the `GRANT` statement; for example, `'jeffrey'@'localhost'`. The user and host parts of the account name correspond to the `user` and `host` column values of the `user` table row for the account.

`DROP USER` as present in MySQL 5.0.0 removes only accounts that have no privileges. In MySQL 5.0.2, it was modified to remove account privileges as well. This means that the procedure for removing an account depends on your version of MySQL.

As of MySQL 5.0.2, you can remove an account and its privileges as follows:

```
DROP USER user;
```

The statement removes privilege rows for the account from all grant tables.

In MySQL 5.0.0 and 5.0.1, `DROP USER` deletes only MySQL accounts that have no privileges. In these MySQL versions, it serves only to remove each account record from the `user` table. To remove a MySQL account completely (including all of its privileges), you should use the following procedure, performing these steps in the order shown:

1. Use `SHOW GRANTS` to determine what privileges the account has. See [Section 13.5.4.12, “SHOW GRANTS Syntax”](#).
2. Use `REVOKE` to revoke the privileges displayed by `SHOW GRANTS`. This removes rows for the account from all the grant tables except the `user` table, and revokes any global privileges listed in the `user` table. See [Section 13.5.1.3, “GRANT Syntax”](#).
3. Delete the account by using `DROP USER` to remove the `user` table record.

Important: `DROP USER` does not automatically close any open user sessions. Rather, in the event that a user with an open session is dropped, the statement does not take effect until that user's session is closed. Once the session is closed, the user is dropped, and that user's next attempt to log in will fail. *This is by*

design.

13.5.1.3. GRANT Syntax

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
  ON [object_type] {tbl_name | * | *.* | db_name.*}
  TO user [IDENTIFIED BY [PASSWORD] 'password']
    [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
  [REQUIRE
    NONE |
    [{SSL| X509}]
    [CIPHER 'cipher' [AND]]
    [ISSUER 'issuer' [AND]]
    [SUBJECT 'subject']]
  [WITH with_option [with_option] ...]
```

```
object_type =
  TABLE
  | FUNCTION
  | PROCEDURE
```

```
with_option =
  GRANT OPTION
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
```

The GRANT statement enables system administrators to create MySQL user accounts and to grant rights to from accounts. To use GRANT, you must have the GRANT OPTION privilege, and you must have the privileges that you are granting. The REVOKE statement is related and enables administrators to remove account privileges. See [Section 13.5.1.5, “REVOKE Syntax”](#).

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in [Chapter 5, Database Administration](#), which you should consult for additional details.

Important: Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. Whenever you update to a new version of MySQL, you should update your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` system variable is set to a non-zero value, `REVOKE` cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (`GRANT` will not create such rows when `lower_case_table_names` is set, but such rows might have been created prior to setting the variable.)

Privileges can be granted at several levels:

- **Global level**

Global privileges apply to all databases on a given server. These privileges are stored in the `mysql.user` table. `GRANT ALL ON *.*` and `REVOKE ALL ON *.*` grant and revoke only global privileges.

- **Database level**

Database privileges apply to all objects in a given database. These privileges are stored in the `mysql.db` and `mysql.host` tables. `GRANT ALL ON db_name.*` and `REVOKE ALL ON db_name.*` grant and revoke only database privileges.

- **Table level**

Table privileges apply to all columns in a given table. These privileges are stored in the `mysql.tables_priv` table. `GRANT ALL ON db_name.tbl_name` and `REVOKE ALL ON db_name.tbl_name` grant and revoke only table privileges.

- **Column level**

Column privileges apply to single columns in a given table. These privileges are stored in the `mysql.columns_priv` table. When using `REVOKE`, you must specify the same columns that were granted.

- **Routine level**

The `CREATE ROUTINE`, `ALTER ROUTINE`, `EXECUTE`, and `GRANT` privileges apply to stored routines (functions and procedures). They can be granted at the global and database levels. Also, except for `CREATE ROUTINE`, these

privileges can be granted at the routine level for individual routines and are stored in the `mysql.procs_priv` table.

The `object_type` clause was added in MySQL 5.0.6. It should be specified as `TABLE`, `FUNCTION`, or `PROCEDURE` when the following object is a table, a stored function, or a stored procedure.

For the `GRANT` and `REVOKE` statements, `priv_type` can be specified as any of the following:

Privilege	Meaning
ALL [PRIVILEGES]	Sets all simple privileges except <code>GRANT OPTION</code>
ALTER	Enables use of <code>ALTER TABLE</code>
ALTER ROUTINE	Enables stored routines to be altered or dropped
CREATE	Enables use of <code>CREATE TABLE</code>
CREATE ROUTINE	Enables creation of stored routines
CREATE TEMPORARY TABLES	Enables use of <code>CREATE TEMPORARY TABLE</code>
CREATE USER	Enables use of <code>CREATE USER</code> , <code>DROP USER</code> , <code>RENAME USER</code> , and <code>REVOKE ALL PRIVILEGES</code> .
CREATE VIEW	Enables use of <code>CREATE VIEW</code>
DELETE	Enables use of <code>DELETE</code>
DROP	Enables use of <code>DROP TABLE</code>
EXECUTE	Enables the user to run stored routines
FILE	Enables use of <code>SELECT ... INTO OUTFILE</code> and <code>LOAD DATA INFILE</code>
INDEX	Enables use of <code>CREATE INDEX</code> and <code>DROP INDEX</code>
INSERT	Enables use of <code>INSERT</code>
LOCK TABLES	Enables use of <code>LOCK TABLES</code> on tables for which you have the <code>SELECT</code> privilege
PROCESS	Enables use of <code>SHOW FULL PROCESSLIST</code>
REFERENCES	Not implemented

RELOAD	Enables use of FLUSH
REPLICATION CLIENT	Enables the user to ask where slave or master servers are
REPLICATION SLAVE	Needed for replication slaves (to read binary log events from the master)
SELECT	Enables use of SELECT
SHOW DATABASES	SHOW DATABASES shows all databases
SHOW VIEW	Enables use of SHOW CREATE VIEW
SHUTDOWN	Enables use of mysqladmin shutdown
SUPER	Enables use of CHANGE MASTER, KILL, PURGE MASTER LOGS, and SET GLOBAL statements, the mysqladmin debug command; allows you to connect (once) even if max_connections is reached
UPDATE	Enables use of UPDATE
USAGE	Synonym for “no privileges”
GRANT OPTION	Enables privileges to be granted

The EXECUTE privilege is not operational until MySQL 5.0.3. CREATE VIEW and SHOW VIEW were added in MySQL 5.0.1. CREATE USER, CREATE ROUTINE, and ALTER ROUTINE were added in MySQL 5.0.3.

The REFERENCES privilege currently is unused.

USAGE can be specified when you want to create a user that has no privileges.

Use SHOW GRANTS to determine what privileges an account has. See [Section 13.5.4.12, “SHOW GRANTS Syntax”](#).

You can assign global privileges by using ON *.* syntax or database-level privileges by using ON db_name.* syntax. If you specify ON * and you have selected a default database, the privileges are granted in that database.

(Warning: If you specify ON * and you have *not* selected a default database, the privileges granted are global.)

The FILE, PROCESS, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, SHUTDOWN, and SUPER privileges are administrative privileges that can

only be granted globally (using `ON *.*` syntax).

Other privileges can be granted globally or at more specific levels.

The *priv_type* values that you can specify for a table are `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT OPTION`, `INDEX`, `ALTER`, `CREATE VIEW` and `SHOW VIEW`.

The *priv_type* values that you can specify for a column (that is, when you use a *column_list* clause) are `SELECT`, `INSERT`, and `UPDATE`.

The *priv_type* values that you can specify at the routine level are `ALTER ROUTINE`, `EXECUTE`, and `GRANT OPTION`. `CREATE ROUTINE` is not a routine-level privilege because you must have this privilege to create a routine in the first place.

For the global, database, table, and routine levels, `GRANT ALL` assigns only the privileges that exist at the level you are granting. For example, `GRANT ALL ON db_name.*` is a database-level statement, so it does not grant any global-only privileges such as `FILE`.

MySQL allows you to grant privileges even on database objects that do not exist. In such cases, the privileges to be granted must include the `CREATE` privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for database objects that are to be created at a later time.

Important: *MySQL does not automatically revoke any privileges when you drop a table or database.* However, if you drop a routine, any routine-level privileges granted for that routine are revoked.

Note: the `'_'` and `'%'` wildcards are allowed when specifying database names in `GRANT` statements that grant privileges at the global or database levels. This means, for example, that if you want to use a `'_'` character as part of a database name, you should specify it as `'_'` in the `GRANT` statement, to prevent the user from being able to access additional databases matching the wildcard pattern; for example, `GRANT ... ON `foo_bar`.* TO ...`

To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *user* value in the form `user_name@host_name`. If a *user_name* or

host_name value is legal as an unquoted identifier, you need not quote it. However, quotes are necessary to specify a *user_name* string containing special characters (such as '-'), or a *host_name* string containing special characters or wildcard characters (such as '%'); for example, 'test-user'@'test-hostname'. Quote the username and hostname separately.

You can specify wildcards in the hostname. For example, *user_name*@'%loc.gov' applies to *user_name* for any host in the loc.gov domain, and *user_name*@'144.155.166.%' applies to *user_name* for any host in the 144.155.166 class C subnet.

The simple form *user_name* is a synonym for *user_name*@'%'.

MySQL does not support wildcards in usernames. Anonymous users are defined by inserting entries with `User=''` into the `mysql.user` table or by creating a user with an empty name with the GRANT statement:

```
GRANT ALL ON test.* TO ''@'localhost' ...
```

When specifying quoted values, quote database, table, column, and routine names as identifiers, using backticks ('`'). Quote hostnames, usernames, and passwords as strings, using single quotes ('').

Warning: If you allow anonymous users to connect to the MySQL server, you should also grant privileges to all local users as *user_name*@localhost. Otherwise, the anonymous user account for localhost in the `mysql.user` table (created during MySQL installation) is used when named users try to log in to the MySQL server from the local machine. For details, see [Section 5.8.5, “Access Control, Stage 1: Connection Verification”](#).

You can determine whether this applies to you by executing the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

If you want to delete the local anonymous user account to avoid the problem just described, use these statements:

```
DELETE FROM mysql.user WHERE Host='localhost' AND User='';  
FLUSH PRIVILEGES;
```

GRANT supports hostnames up to 60 characters long. Database, table, column, and routine names can be up to 64 characters. Usernames can be up to 16 characters. **Note:** *The allowable length for usernames cannot be changed by altering the `mysql.user` table, and attempting to do so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server.* You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure prescribed by MySQL AB that is described in [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

The privileges for a table, column, or routine are formed additively as the logical OR of the privileges at each of the privilege levels. For example, if the `mysql.user` table specifies that a user has a global SELECT privilege, the privilege cannot be denied by an entry at the database, table, or column level.

The privileges for a column can be calculated as follows:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

In most cases, you grant rights to a user at only one of the privilege levels, so life is not normally this complicated. The details of the privilege-checking procedure are presented in [Section 5.8, “The MySQL Access Privilege System”](#).

If you grant privileges for a username/hostname combination that does not exist in the `mysql.user` table, an entry is added and remains there until deleted with a DELETE statement. In other words, GRANT may create user table entries, but REVOKE does not remove them; you must do that explicitly using DROP USER or DELETE.

Warning: If you create a new user but do not specify an IDENTIFIED BY clause, the user has no password. This is very insecure. As of MySQL 5.0.2, you can enable the NO_AUTO_CREATE_USER SQL mode to prevent GRANT from creating a new user if it would otherwise do so, unless IDENTIFIED BY is given to provide the new user a non-empty password.

If a new user is created or if you have global grant privileges, the user's password is set to the password specified by the IDENTIFIED BY clause, if one is

given. If the user already had a password, this is replaced by the new one.

Passwords can also be set with the `SET PASSWORD` statement. See [Section 13.5.1.6, “SET PASSWORD Syntax”](#).

In the `IDENTIFIED BY` clause, the password should be given as the literal password value. It is unnecessary to use the `PASSWORD()` function as it is for the `SET PASSWORD` statement. For example:

```
GRANT ... IDENTIFIED BY 'mypass';
```

If you do not want to send the password in clear text and you know the hashed value that `PASSWORD()` would return for the password, you can specify the hashed value preceded by the keyword `PASSWORD`:

```
GRANT ...  
IDENTIFIED BY PASSWORD '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

In a C program, you can get the hashed value by using the `make_scrambled_password()` C API function.

If you grant privileges for a database, an entry in the `mysql.db` table is created if needed. If all privileges for the database are removed with `REVOKE`, this entry is deleted.

The `SHOW DATABASES` privilege enables the account to see database names by issuing the `SHOW DATABASE` statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the `--skip-show-database` option.

If a user has no privileges for a table, the table name is not displayed when the user requests a list of tables (for example, with a `SHOW TABLES` statement).

The `WITH GRANT OPTION` clause gives the user the ability to give to other users any privileges the user has at the specified privilege level. You should be careful to whom you give the `GRANT OPTION` privilege, because two users with different privileges may be able to join privileges!

You cannot grant another user a privilege which you yourself do not have; the `GRANT OPTION` privilege enables you to assign only those privileges which you yourself possess.

Be aware that when you grant a user the `GRANT OPTION` privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the `INSERT` privilege on a database. If you then grant the `SELECT` privilege on the database and specify `WITH GRANT OPTION`, that user can give to other users not only the `SELECT` privilege, but also `INSERT`. If you then grant the `UPDATE` privilege to the user on the database, the user can grant `INSERT`, `SELECT`, and `UPDATE`.

For a non-administrative user, you should not grant the `ALTER` privilege globally or for the `mysql` database. If you do that, the user can try to subvert the privilege system by renaming tables!

The `MAX_QUERIES_PER_HOUR` count, `MAX_UPDATES_PER_HOUR` count, and `MAX_CONNECTIONS_PER_HOUR` count options limit the number of queries, updates, and logins a user can perform during any given one-hour period. If *count* is 0 (the default), this means that there is no limitation for that user.

The `MAX_USER_CONNECTIONS` count option, implemented in MySQL 5.0.3, limits the maximum number of simultaneous connections that the account can make. If *count* is 0 (the default), the `max_user_connections` system variable determines the number of simultaneous connections for the account.

Note: To specify any of these resource-limit options for an existing user without affecting existing privileges, use `GRANT USAGE ON *.* ... WITH MAX_...`

See [Section 5.9.4, “Limiting Account Resources”](#).

MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the username and password. To specify SSL-related options for a MySQL account, use the `REQUIRE` clause of the `GRANT` statement. (For background information on the use of SSL with MySQL, see [Section 5.9.7, “Using Secure Connections”](#).)

There are a number of different possibilities for limiting connection types for a given account:

- If the account has no SSL or X509 requirements, unencrypted connections are allowed if the username and password are valid. However, encrypted connections can also be used, at the client's option, if the client has the

proper certificate and key files.

- The `REQUIRE SSL` option tells the server to allow only SSL-encrypted connections for the account. Note that this option can be omitted if there are any access-control rows that allow non-SSL connections.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
  IDENTIFIED BY 'goodsecret' REQUIRE SSL;
```

- `REQUIRE X509` means that the client must have a valid certificate but that the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
  IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

- `REQUIRE ISSUER 'issuer'` places the restriction on connection attempts that the client must present a valid X509 certificate issued by CA `'issuer'`. If the client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
  IDENTIFIED BY 'goodsecret'  
  REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/  
  O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
```

Note that the `'issuer'` value should be entered as a single string.

- `REQUIRE SUBJECT 'subject'` places the restriction on connection attempts that the client must present a valid X509 certificate containing the subject *subject*. If the client presents a certificate that is valid but has a different subject, the server rejects the connection.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'  
  IDENTIFIED BY 'goodsecret'  
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/  
  O=MySQL demo client certificate/  
  CN=Tonu Samuel/Email=tonu@example.com';
```

Note that the `'subject'` value should be entered as a single string.

- REQUIRE CIPHER 'cipher' is needed to ensure that ciphers and key lengths of sufficient strength are used. SSL itself can be weak if old algorithms using short encryption keys are used. Using this option, you can ask that a specific cipher method is used to allow a connection.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The SUBJECT, ISSUER, and CIPHER options can be combined in the REQUIRE clause like this:

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
  O=MySQL demo client certificate/
  CN=Tonu Samuel/Email=tonu@example.com'
  AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
  O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The AND keyword is optional between REQUIRE options.

The order of the options does not matter, but no option can be specified twice.

When **mysqld** starts, all privileges are read into memory. For details, see [Section 5.8.7, “When Privilege Changes Take Effect”](#).

Note that if you are using table, column, or routine privileges for even one user, the server examines table, column, and routine privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

The biggest differences between the standard SQL and MySQL versions of GRANT are:

- In MySQL, privileges are associated with the combination of a hostname and username and not with only a username.
- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.

- MySQL does not support the standard SQL TRIGGER or UNDER privileges.
- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges the user has been granted are revoked. This is also true in MySQL 5.0.2 and up if you use DROP USER. Before 5.0.2, the granted privileges are not automatically revoked; you must revoke them yourself. See [Section 13.5.1.2, “DROP USER Syntax”](#).
- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped only with explicit REVOKE statements or by manipulating values stored in the MySQL grant tables.
- In MySQL, it is possible to have the INSERT privilege for only some of the columns in a table. In this case, you can still execute INSERT statements on the table, provided that you omit those columns for which you do not have the INSERT privilege. The omitted columns are set to their implicit default values if strict SQL mode is not enabled. In strict mode, the statement is rejected if any of the omitted columns have no default value. (Standard SQL requires you to have the INSERT privilege on all columns.) [Section 5.2.5, “The Server SQL Mode”](#), discusses strict mode. [Section 11.1.4, “Data Type Default Values”](#), discusses implicit default values.

13.5.1.4. RENAME USER Syntax

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

The RENAME USER statement renames existing MySQL accounts. To use it, you must have the global CREATE USER privilege or the UPDATE privilege for the mysql database. An error occurs if any old account does not exist or any new account exists. Each account is named using the same format as for the GRANT statement; for example, 'jeffrey'@'localhost'. The user and host parts of the account name correspond to the User and Host column values of the user table row for the account.

The RENAME USER statement was added in MySQL 5.0.2.

13.5.1.5. REVOKE Syntax

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list))] ...  
    ON [object_type] {tbl_name | * | *.* | db_name.*}  
    FROM user [, user] ...
```

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

The REVOKE statement enables system administrators to revoke privileges from MySQL accounts. To use REVOKE, you must have the GRANT OPTION privilege, and you must have the privileges that you are revoking.

For details on the levels at which privileges exist, the allowable *priv_type* values, and the syntax for specifying users and passwords, see [Section 13.5.1.3, “GRANT Syntax”](#)

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` system variable is set to a non-zero value, REVOKE cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (GRANT will not create such rows when `lower_case_table_names` is set, but such rows might have been created prior to setting the variable.)

To revoke all privileges, use the following syntax, which drops all global, database-, table-, and column-level privileges for the named user or users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this REVOKE syntax, you must have the global CREATE USER privilege or the UPDATE privilege for the `mysql` database.

13.5.1.6. SET PASSWORD Syntax

```
SET PASSWORD [FOR user] = PASSWORD('some password')
```

The SET PASSWORD statement assigns a password to an existing MySQL user account.

With no FOR clause, this statement sets the password for the current user. Any client that has connected to the server using a non-anonymous account can change the password for that account.

With a FOR clause, this statement sets the password for a specific account on the current server host. Only clients that have the UPDATE privilege for the mysql database can do this. The *user* value should be given in *user_name@host_name* format, where *user_name* and *host_name* are exactly as they are listed in the User and Host columns of the mysql.user table entry. For example, if you had an entry with User and Host column values of 'bob' and '%.loc.gov', you would write the statement like this:

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

That is equivalent to the following statements:

```
UPDATE mysql.user SET Password=PASSWORD('newpass')
  WHERE User='bob' AND Host='%.loc.gov';
FLUSH PRIVILEGES;
```

Note: If you are connecting to a MySQL 4.1 or later server using a pre-4.1 client program, do not use the preceding SET PASSWORD or UPDATE statement without reading [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#), first. The password format changed in MySQL 4.1, and under certain circumstances it is possible that if you change your password, you might not be able to connect to the server afterward.

You can see which account the server authenticated you as by executing SELECT CURRENT_USER().

13.5.2. Table Maintenance Statements

13.5.2.1. ANALYZE TABLE Syntax

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

ANALYZE TABLE analyzes and stores the key distribution for a table. During the analysis, the table is locked with a read lock for MyISAM and BDB. For InnoDB the table is locked with a write lock. This statement works with MyISAM, BDB, and InnoDB tables. For MyISAM tables, this statement is equivalent to using **myisamchk --analyze**.

For more information on how the analysis works within InnoDB, see [Section 14.2.16, “Restrictions on InnoDB Tables”](#).

MySQL uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant.

This statement requires `SELECT` and `INSERT` privileges for the table.

`ANALYZE TABLE` returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always analyze
Msg_type	One of status, error, info, or warning
Msg_text	The message

You can check the stored key distribution with the `SHOW INDEX` statement. See [Section 13.5.4.13, “SHOW INDEX Syntax”](#).

If the table has not changed since the last `ANALYZE TABLE` statement, the table is not analyzed again.

`ANALYZE TABLE` statements are written to the binary log unless the optional `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`) is used. This is done so that `ANALYZE TABLE` statements used on a MySQL server acting as a replication master will be replicated by default to the replication slave.

13.5.2.2. `BACKUP TABLE` Syntax

```
BACKUP TABLE tbl_name [, tbl_name] ... TO '/path/to/backup/directory
```

Note: This statement is deprecated. We are working on a better replacement for it that will provide online backup capabilities. In the meantime, the **mysqlhotcopy** script can be used instead.

`BACKUP TABLE` copies to the backup directory the minimum number of table files needed to restore the table, after flushing any buffered changes to disk. The statement works only for MyISAM tables. It copies the `.frm` definition and `.MYD` data files. The `.MYI` index file can be rebuilt from those two files. The directory should be specified as a full pathname. To restore the table, use `RESTORE TABLE`.

During the backup, a read lock is held for each table, one at time, as they are being backed up. If you want to back up several tables as a snapshot (preventing any of them from being changed during the backup operation), issue a LOCK TABLES statement first, to obtain a read lock for all tables in the group.

BACKUP TABLE returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always backup
Msg_type	One of status, error, info, or warning
Msg_text	The message

13.5.2.3. CHECK TABLE Syntax

CHECK TABLE *tbl_name* [, *tbl_name*] ... [*option*] ...

option = {FOR UPGRADE | QUICK | FAST | MEDIUM | EXTENDED | CHANGED}

CHECK TABLE checks a table or tables for errors. CHECK TABLE works for MyISAM, InnoDB, and (as of MySQL 5.0.16) ARCHIVE tables. For MyISAM tables, the key statistics are updated as well.

As of MySQL 5.0.2, CHECK TABLE can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

CHECK TABLE returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always check
Msg_type	One of status, error, info, or warning
Msg_text	The message

Note that the statement might produce many rows of information for each checked table. The last row has a Msg_type value of status and the Msg_text normally should be OK. If you don't get OK, or Table is already up to date

you should normally run a repair of the table. See [Section 5.10.4, “Table Maintenance and Crash Recovery”](#). Table is already up to date means that the storage engine for the table indicated that there was no need to check the table.

The `FOR UPGRADE` option checks whether the named tables are compatible with the current version of MySQL. This option was added in MySQL 5.0.19. With `FOR UPGRADE`, the server checks each table to determine whether there have been any incompatible changes in any of the table's data types or indexes since the table was created. If not, the check succeeds. Otherwise, if there is a possible incompatibility, the server runs a full check on the table (which might take some time). If the full check succeeds, the server marks the table's `.frm` file with the current MySQL version number. Marking the `.frm` file ensures that further checks for the table with the same version of the server will be fast.

Incompatibilities might occur because the storage format for a data type has changed or because its sort order has changed. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.

Currently, `FOR UPGRADE` discovers these incompatibilities:

- The indexing order for end-space in `TEXT` columns for `InnoDB` and `MyISAM` tables changed between MySQL 4.1 and 5.0.
- The storage method of the new `DECIMAL` data type changed between MySQL 5.0.3 and 5.0.5.

The other check options that can be given are shown in the following table. These options apply only to checking `MyISAM` tables and are ignored for `InnoDB` tables and views.

Type	Meaning
QUICK	Do not scan the rows to check for incorrect links.
FAST	Check only tables that have not been closed properly.
CHANGED	Check only tables that have been changed since the last check or that have not been closed properly.
	Scan rows to verify that deleted links are valid. This also calculates a

MEDIUM	key checksum for the rows and verifies this with a calculated checksum for the keys.
EXTENDED	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time.

If none of the options `QUICK`, `MEDIUM`, or `EXTENDED` are specified, the default check type for dynamic-format MyISAM tables is `MEDIUM`. This has the same result as running `myisamchk --medium-check tbl_name` on the table. The default check type also is `MEDIUM` for static-format MyISAM tables, unless `CHANGED` or `FAST` is specified. In that case, the default is `QUICK`. The row scan is skipped for `CHANGED` and `FAST` because the rows are very seldom corrupted.

You can combine check options, as in the following example that does a quick check on the table to determine whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

Note: In some cases, `CHECK TABLE` changes the table. This happens if the table is marked as “corrupted” or “not closed properly” but `CHECK TABLE` does not find any problems in the table. In this case, `CHECK TABLE` marks the table as okay.

If a table is corrupted, it is most likely that the problem is in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

If you just want to check a table that you assume is okay, you should use no check options or the `QUICK` option. The latter should be used when you are in a hurry and can take the very small risk that `QUICK` does not find an error in the data file. (In most cases, under normal usage, MySQL should find any error in the data file. If this happens, the table is marked as “corrupted” and cannot be used until it is repaired.)

`FAST` and `CHANGED` are mostly intended to be used from a script (for example, to be executed from `cron`) if you want to check tables from time to time. In most cases, `FAST` is to be preferred over `CHANGED`. (The only case when it is not preferred is when you suspect that you have found a bug in the MyISAM code.)

`EXTENDED` is to be used only after you have run a normal check but still get

strange errors from a table when MySQL tries to update a row or find a row by key. This is very unlikely if a normal check has succeeded.

Some problems reported by `CHECK TABLE` cannot be corrected automatically:

- Found row where the `auto_increment` column has the value 0.

This means that you have a row in the table where the `AUTO_INCREMENT` index column contains the value 0. (It is possible to create a row where the `AUTO_INCREMENT` column is 0 by explicitly setting the column to 0 with an `UPDATE` statement.)

This is not an error in itself, but could cause trouble if you decide to dump the table and restore it or do an `ALTER TABLE` on the table. In this case, the `AUTO_INCREMENT` column changes value according to the rules of `AUTO_INCREMENT` columns, which could cause problems such as a duplicate-key error.

To get rid of the warning, simply execute an `UPDATE` statement to set the column to some value other than 0.

13.5.2.4. `CHECKSUM TABLE` Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

`CHECKSUM TABLE` reports a table checksum.

With `QUICK`, the live table checksum is reported if it is available, or `NULL` otherwise. This is very fast. A live checksum is enabled by specifying the `CHECKSUM=1` table option when you create the table; currently, this is supported only for `MyISAM` tables. See [Section 13.1.5, “CREATE TABLE Syntax”](#).

With `EXTENDED`, the entire table is read row by row and the checksum is calculated. This can be very slow for large tables.

If neither `QUICK` nor `EXTENDED` is specified, MySQL returns a live checksum if the table storage engine supports it and scans the table otherwise.

For a non-existent table, `CHECKSUM TABLE` returns `NULL` and, as of MySQL 5.0.3, generates a warning.

13.5.2.5. OPTIMIZE TABLE Syntax

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ..
```

OPTIMIZE TABLE should be used if you have deleted a large part of a table or if you have made many changes to a table with variable-length rows (tables that have VARCHAR, VARBINARY, BLOB, or TEXT columns). Deleted rows are maintained in a linked list and subsequent INSERT operations reuse old row positions. You can use OPTIMIZE TABLE to reclaim the unused space and to defragment the data file.

This statement requires SELECT and INSERT privileges for the table.

In most setups, you need not run OPTIMIZE TABLE at all. Even if you do a lot of updates to variable-length rows, it is not likely that you need to do this more than once a week or month and only on certain tables.

OPTIMIZE TABLE works only for MyISAM, BDB, and InnoDB tables.

For MyISAM tables, OPTIMIZE TABLE works as follows:

1. If the table has deleted or split rows, repair the table.
2. If the index pages are not sorted, sort them.
3. If the table's statistics are not up to date (and the repair could not be accomplished by sorting the index), update them.

For BDB tables, OPTIMIZE TABLE currently is mapped to ANALYZE TABLE. See [Section 13.5.2.1, “ANALYZE TABLE Syntax”](#).

For InnoDB tables, OPTIMIZE TABLE is mapped to ALTER TABLE, which rebuilds the table to update index statistics and free unused space in the clustered index.

You can make OPTIMIZE TABLE work on other storage engines by starting **mysqld** with the `--skip-new` or `--safe-mode` option. In this case, OPTIMIZE TABLE is just mapped to ALTER TABLE.

OPTIMIZE TABLE returns a result set with the following columns:

--

Column	Value
Table	The table name
Op	Always optimize
Msg_type	One of status, error, info, or warning
Msg_text	The message

Note that MySQL locks the table during the time `OPTIMIZE TABLE` is running.

`OPTIMIZE TABLE` statements are written to the binary log unless the optional `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`) is used. This is done so that `OPTIMIZE TABLE` statements used on a MySQL server acting as a replication master will be replicated by default to the replication slave.

13.5.2.6. REPAIR TABLE Syntax

```
REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE
      tbl_name [, tbl_name] ... [QUICK] [EXTENDED] [USE_FRM]
```

`REPAIR TABLE` repairs a possibly corrupted table. By default, it has the same effect as `myisamchk --recover tbl_name`. `REPAIR TABLE` works for MyISAM and for ARCHIVE tables. See [Section 14.1, “The MyISAM Storage Engine”](#), and [Section 14.8, “The ARCHIVE Storage Engine”](#).

This statement requires `SELECT` and `INSERT` privileges for the table.

Normally, you should never have to run this statement. However, if disaster strikes, `REPAIR TABLE` is very likely to get back all your data from a MyISAM table. If your tables become corrupted often, you should try to find the reason for it, to eliminate the need to use `REPAIR TABLE`. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#), and [Section 14.1.4, “MyISAM Table Problems”](#).

Warning: If the server dies during a `REPAIR TABLE` operation, it is essential after restarting it that you immediately execute another `REPAIR TABLE` statement for the table before performing any other operations on it. (It is always a good idea to start by making a backup.) In the worst case, you might have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely but possible scenario.

REPAIR TABLE returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always repair
Msg_type	One of status, error, info, or warning
Msg_text	The message

The REPAIR TABLE statement might produce many rows of information for each repaired table. The last row has a Msg_type value of status and Msg_text normally should be OK. If you do not get OK, you should try repairing the table with **myisamchk --safe-recover**. (REPAIR TABLE does not yet implement all the options of **myisamchk**.) With **myisamchk --safe-recover**, you can also use options that REPAIR TABLE does not support, such as `--max-record-length`.

If QUICK is given, REPAIR TABLE tries to repair only the index tree. This type of repair is like that done by **myisamchk --recover --quick**.

If you use EXTENDED, MySQL creates the index row by row instead of creating one index at a time with sorting. This type of repair is like that done by **myisamchk --safe-recover**.

There is also a USE_FRM mode available for REPAIR TABLE. Use this if the .MYI index file is missing or if its header is corrupted. In this mode, MySQL re-creates the .MYI file using information from the .frm file. This kind of repair cannot be done with **myisamchk**. **Note:** Use this mode *only* if you cannot use regular REPAIR modes. The .MYI header contains important table metadata (in particular, current AUTO_INCREMENT value and Delete link) that are lost in REPAIR ... USE_FRM. Don't use USE_FRM if the table is compressed because this information is also stored in the .MYI file.

REPAIR TABLE statements are written to the binary log unless the optional NO_WRITE_TO_BINLOG keyword (or its alias LOCAL) is used. This is done so that REPAIR TABLE statements used on a MySQL server acting as a replication master will be replicated by default to the replication slave.

13.5.2.7. RESTORE TABLE Syntax

```
RESTORE TABLE tbl_name [, tbl_name] ... FROM '/path/to/backup/direct
```

RESTORE TABLE restores the table or tables from a backup that was made with BACKUP TABLE. Existing tables are not overwritten; if you try to restore over an existing table, an error occurs. Just as for BACKUP TABLE, RESTORE TABLE currently works only for MyISAM tables. The directory should be specified as a full pathname.

The backup for each table consists of its .frm format file and .MYD data file. The restore operation restores those files, and then uses them to rebuild the .MYI index file. Restoring takes longer than backing up due to the need to rebuild the indexes. The more indexes the table has, the longer it takes.

RESTORE TABLE returns a result set with the following columns:

Column	Value
Table	The table name
Op	Always restore
Msg_type	One of status, error, info, or warning
Msg_text	The message

13.5.3. SET Syntax

```
SET variable_assignment [, variable_assignment] ...
```

variable_assignment:

```
    user_var_name = expr  
    | [GLOBAL | SESSION] system_var_name = expr  
    | [@@global. | @@session. | @@] system_var_name = expr
```

The SET statement assigns values to different types of variables that affect the operation of the server or your client. Older versions of MySQL employed SET OPTION, but this syntax is deprecated in favor of SET without OPTION.

This section describes use of SET for assigning values to system variables or user variables. For general information about these types of variables, see [Section 5.2.2, “Server System Variables”](#), and [Section 9.3, “User-Defined Variables”](#). System variables also can be set at server startup, as described in [Section 5.2.3, “Using System Variables”](#).

Some variants of SET syntax are used in other contexts:

- SET PASSWORD assigns account passwords. See [Section 13.5.1.6, “SET PASSWORD Syntax”](#).
- SET TRANSACTION ISOLATION LEVEL sets the isolation level for transaction processing. See [Section 13.4.6, “SET TRANSACTION Syntax”](#).
- SET is used within stored routines to assign values to local routine variables. See [Section 17.2.7.2, “Variable SET Statement”](#).

The following discussion shows the different SET syntaxes that you can use to set variables. The examples use the = assignment operator, but the := operator also is allowable.

A user variable is written as @var_name and can be set as follows:

```
SET @var_name = expr;
```

Many system variables are dynamic and can be changed while the server runs by using the SET statement. For a list, see [Section 5.2.3.2, “Dynamic System Variables”](#). To change a system variable with SET, refer to it as var_name, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by GLOBAL or @@global.. The SUPER privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by SESSION, @@session., or @@. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- LOCAL and @@local. are synonyms for SESSION and @@session..
- If no modifier is present, SET changes the session variable.

A SET statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent GLOBAL or SESSION modifier in the statement is used for following variables that have no modifier

specified.

Examples:

```
SET sort_buffer_size=10000;  
SET @@local.sort_buffer_size=10000;  
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=100000  
SET @@sort_buffer_size=1000000;  
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000
```

When you assign a value to a system variable with SET, you cannot use suffix letters in the value (as can be done with startup options). However, the value can take the form of an expression:

```
SET sort_buffer_size = 10 * 1024 * 1024;
```

The @@var_name syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the SET GLOBAL statement).

To prevent incorrect usage, MySQL produces an error if you use SET GLOBAL with a variable that can only be used with SET SESSION or if you do not specify GLOBAL (or @@global.) when setting a global variable.

To set a SESSION variable to the GLOBAL value or a GLOBAL value to the compiled-in MySQL default value, use the DEFAULT keyword. For example, the following two statements are identical in setting the session value of max_join_size to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to DEFAULT. In such cases, use of DEFAULT results in an error.

You can refer to the values of specific global or session system variables in expressions by using one of the @@-modifiers. For example, you can retrieve values in a SELECT statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as @@var_name (that is, when you do not specify @@global. or @@session.), MySQL returns the session value if it exists and the global value otherwise. (This differs from SET @@var_name = value, which always refers to the session value.)

To display system variables names and values, use the SHOW VARIABLES statement. (See [Section 13.5.4.24, “SHOW VARIABLES Syntax”](#).)

The following list describes options that have non-standard syntax or that are not described in the list of system variables found in [Section 5.2.2, “Server System Variables”](#). Although the options described here are not displayed by SHOW VARIABLES, you can obtain their values with SELECT (with the exception of CHARACTER SET and SET NAMES). For example:

```
mysql> SELECT @@AUTOCOMMIT;
+-----+
| @@AUTOCOMMIT |
+-----+
|           1 |
+-----+
```

The lettercase of these options does not matter.

- AUTOCOMMIT = {0 | 1}

Set the autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0 you have to use COMMIT to accept a transaction or ROLLBACK to cancel it. By default, client connections begin with AUTOCOMMIT set to 1. If you change AUTOCOMMIT mode from 0 to 1, MySQL performs an automatic COMMIT of any open transaction. Another way to begin a

transaction is to use a `START TRANSACTION` or `BEGIN` statement. See [Section 13.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

- `BIG_TABLES = {0 | 1}`

If set to 1, all temporary tables are stored on disk rather than in memory. This is a little slower, but the error `The table tbl_name is full` does not occur for `SELECT` operations that require a large temporary table. The default value for a new connection is 0 (use in-memory temporary tables). Normally, you should never need to set this variable, because in-memory tables are automatically converted to disk-based tables as required. (**Note:** This variable was formerly named `SQL_BIG_TABLES`.)

- `CHARACTER SET {charset_name | DEFAULT}`

This maps all strings from and to the client with the given mapping. You can add new mappings by editing `sql/convert.cc` in the MySQL source distribution. `SET CHARACTER SET` sets three session system variables: `character_set_client` and `character_set_results` are set to the given character set, and `character_set_connection` to the value of `character_set_database`. See [Section 10.4, “Connection Character Sets and Collations”](#).

The default mapping can be restored by using the value `DEFAULT`. The default depends on the server configuration.

Note that the syntax for `SET CHARACTER SET` differs from that for setting most other options.

- `FOREIGN_KEY_CHECKS = {0 | 1}`

If set to 1 (the default), foreign key constraints for InnoDB tables are checked. If set to 0, they are ignored. Disabling foreign key checking can be useful for reloading InnoDB tables in an order different from that required by their parent/child relationships. See [Section 14.2.6.4, “FOREIGN KEY Constraints”](#).

- `IDENTITY = value`

This variable is a synonym for the `LAST_INSERT_ID` variable. It exists for

compatibility with other database systems. You can read its value with `SELECT @@IDENTITY`, and set it using `SET IDENTITY`.

- `INSERT_ID = value`

Set the value to be used by the following `INSERT` or `ALTER TABLE` statement when inserting an `AUTO_INCREMENT` value. This is mainly used with the binary log.

- `LAST_INSERT_ID = value`

Set the value to be returned from `LAST_INSERT_ID()`. This is stored in the binary log when you use `LAST_INSERT_ID()` in a statement that updates a table. Setting this variable does not update the value returned by the `mysql_insert_id()` C API function.

- `NAMES {'charset_name' [COLLATE 'collation_name']} | DEFAULT}`

`SET NAMES` sets the three session system variables `character_set_client`, `character_set_connection`, and `character_set_results` to the given character set. Setting `character_set_connection` to `charset_name` also sets `collation_connection` to the default collation for `charset_name`. The optional `COLLATE` clause may be used to specify a collation explicitly. See [Section 10.4, “Connection Character Sets and Collations”](#).

The default mapping can be restored by using a value of `DEFAULT`. The default depends on the server configuration.

Note that the syntax for `SET NAMES` differs from that for setting most other options.

- `ONE_SHOT`

This option is a modifier, not a variable. It can be used to influence the effect of variables that set the character set, the collation, and the time zone. `ONE_SHOT` is primarily used for replication purposes: **mysqlbinlog** uses `SET ONE_SHOT` to modify temporarily the values of character set, collation, and time zone variables to reflect at rollforward what they were originally. `ONE_SHOT` is available as of MySQL 5.0.

You cannot use `ONE_SHOT` with other than the allowed set of variables; if you try, you get an error like this:

```
mysql> SET ONE_SHOT max_allowed_packet = 1;
ERROR 1382 (HY000): The 'SET ONE_SHOT' syntax is reserved for pu
internal to the MySQL server
```

If `ONE_SHOT` is used with the allowed variables, it changes the variables as requested, but only for the next non-`SET` statement. After that, the server resets all character set, collation, and time zone-related system variables to their previous values. Example:

```
mysql> SET ONE_SHOT character_set_connection = latin5;
mysql> SET ONE_SHOT collation_connection = latin5_turkish_ci;
```

```
mysql> SHOW VARIABLES LIKE '%_connection';
+-----+-----+
| Variable_name          | Value          |
+-----+-----+
| character_set_connection | latin5         |
| collation_connection    | latin5_turkish_ci |
+-----+-----+
```

```
mysql> SHOW VARIABLES LIKE '%_connection';
+-----+-----+
| Variable_name          | Value          |
+-----+-----+
| character_set_connection | latin1         |
| collation_connection    | latin1_swedish_ci |
+-----+-----+
```

- `SQL_AUTO_IS_NULL = {0 | 1}`

If set to 1 (the default), you can find the last inserted row for a table that contains an `AUTO_INCREMENT` column by using the following construct:

```
WHERE auto_increment_column IS NULL
```

This behavior is used by some ODBC programs, such as Access.

- `SQL_BIG_SELECTS = {0 | 1}`

If set to 0, MySQL aborts `SELECT` statements that are likely to take a very long time to execute (that is, statements for which the optimizer estimates

that the number of examined rows exceeds the value of `max_join_size`). This is useful when an inadvisable `WHERE` statement has been issued. The default value for a new connection is 1, which allows all `SELECT` statements.

If you set the `max_join_size` system variable to a value other than `DEFAULT`, `SQL_BIG_SELECTS` is set to 0.

- `SQL_BUFFER_RESULT = {0 | 1}`

If set to 1, `SQL_BUFFER_RESULT` forces results from `SELECT` statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. The default value is 0.

- `SQL_LOG_BIN = {0 | 1}`

If set to 0, no logging is done to the binary log for the client. The client must have the `SUPER` privilege to set this option. The default value is 1.

- `SQL_LOG_OFF = {0 | 1}`

If set to 1, no logging is done to the general query log for this client. The client must have the `SUPER` privilege to set this option. The default value is 0.

- `SQL_LOG_UPDATE = {0 | 1}`

This variable is deprecated, and is mapped to `SQL_LOG_BIN`.

- `SQL_NOTES = {0 | 1}`

If set to 1 (the default), warnings of `Note` level are recorded. If set to 0, `Note` warnings are suppressed. **mysqldump** includes output to set this variable to 0 so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation. `SQL_NOTES` was added in MySQL 5.0.3.

- `SQL_QUOTE_SHOW_CREATE = {0 | 1}`

If set to 1 (the default), the server quotes identifiers for `SHOW CREATE TABLE`

and `SHOW CREATE DATABASE` statements. If set to 0, quoting is disabled. This option is enabled by default so that replication works for identifiers that require quoting. See [Section 13.5.4.6, “SHOW CREATE TABLE Syntax”](#), and [Section 13.5.4.4, “SHOW CREATE DATABASE Syntax”](#).

- `SQL_SAFE_UPDATES = {0 | 1}`

If set to 1, MySQL aborts `UPDATE` or `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause. This makes it possible to catch `UPDATE` or `DELETE` statements where keys are not used properly and that would probably change or delete a large number of rows. The default value is 0.

- `SQL_SELECT_LIMIT = {value | DEFAULT}`

The maximum number of rows to return from `SELECT` statements. The default value for a new connection is “unlimited.” If you have changed the limit, the default value can be restored by using a `SQL_SELECT_LIMIT` value of `DEFAULT`.

If a `SELECT` has a `LIMIT` clause, the `LIMIT` takes precedence over the value of `SQL_SELECT_LIMIT`.

`SQL_SELECT_LIMIT` does not apply to `SELECT` statements executed within stored routines. It also does not apply to `SELECT` statements that do not produce a result set to be returned to the client. These include `SELECT` statements in subqueries, `CREATE TABLE ... SELECT`, and `INSERT INTO ... SELECT`.

- `SQL_WARNINGS = {0 | 1}`

This variable controls whether single-row `INSERT` statements produce an information string if warnings occur. The default is 0. Set the value to 1 to produce an information string.

- `TIMESTAMP = {timestamp_value | DEFAULT}`

Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. `timestamp_value` should be a Unix epoch timestamp, not a MySQL timestamp.

SET `TIMESTAMP` affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`. The server can be started with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`, in which case SET `TIMESTAMP` affects both functions.

- `UNIQUE_CHECKS = {0 | 1}`

If set to 1 (the default), uniqueness checks for secondary indexes in InnoDB tables are performed. If set to 0, storage engines are allowed to assume that duplicate keys are not present in input data. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to InnoDB.

Note that setting this variable to 0 does not *require* storage engines to ignore duplicate keys. An engine is still allowed to check for them and issue duplicate-key errors if it detects them.

13.5.4. SHOW Syntax

SHOW has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
SHOW CREATE DATABASE db_name
SHOW CREATE FUNCTION funcname
SHOW CREATE PROCEDURE procname
SHOW CREATE TABLE tbl_name
SHOW DATABASES [LIKE 'pattern']
SHOW ENGINE engine_name {LOGS | STATUS }
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW FUNCTION STATUS [LIKE 'pattern']
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW PROCEDURE STATUS [LIKE 'pattern']
SHOW [BDB] LOGS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']
SHOW TRIGGERS
```

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
SHOW WARNINGS [LIMIT [offset,] row_count]
```

The SHOW statement also has forms that provide information about replication master and slave servers and are described in [Section 13.6, “Replication Statements”](#):

```
SHOW BINARY LOGS
SHOW BINLOG EVENTS
SHOW MASTER STATUS
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
```

If the syntax for a given SHOW statement includes a LIKE 'pattern' part, 'pattern' is a string that can contain the SQL '%' and '_' wildcard characters. The pattern is useful for restricting statement output to matching values.

Several SHOW statements also accept a WHERE clause that provides more flexibility in specifying which rows to display. See [Section 20.18, “Extensions to SHOW Statements”](#).

13.5.4.1. SHOW CHARACTER SET Syntax

```
SHOW CHARACTER SET [LIKE 'pattern']
```

The SHOW CHARACTER SET statement shows all available character sets. It takes an optional LIKE clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European | latin1_swedish_ci | 1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1 |
| latin5  | ISO 8859-9 Turkish | latin5_turkish_ci | 1 |
| latin7  | ISO 8859-13 Baltic | latin7_general_ci | 1 |
+-----+-----+-----+-----+
```

The Maxlen column shows the maximum number of bytes required to store one character.

13.5.4.2. SHOW COLLATION Syntax

```
SHOW COLLATION [LIKE 'pattern']
```

The output from `SHOW COLLATION` includes all available character sets. It takes an optional `LIKE` clause whose *pattern* indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-----+-----+-----+-----+-----+-----+
| Collation          | Charset | Id  | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+-----+
| latin1_german1_ci  | latin1  | 5   |         |         | 0       |
| latin1_swedish_ci  | latin1  | 8   | Yes     | Yes     | 0       |
| latin1_danish_ci   | latin1  | 15  |         |         | 0       |
| latin1_german2_ci  | latin1  | 31  |         | Yes     | 2       |
| latin1_bin         | latin1  | 47  |         | Yes     | 0       |
| latin1_general_ci  | latin1  | 48  |         |         | 0       |
| latin1_general_cs  | latin1  | 49  |         |         | 0       |
| latin1_spanish_ci  | latin1  | 94  |         |         | 0       |
+-----+-----+-----+-----+-----+-----+
```

The `Default` column indicates whether a collation is the default for its character set. `Compiled` indicates whether the character set is compiled into the server. `Sortlen` is related to the amount of memory required to sort strings expressed in the character set.

13.5.4.3. SHOW COLUMNS Syntax

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
```

`SHOW COLUMNS` displays information about the columns in a given table. It also works for views as of MySQL 5.0.1.

If the data types differ from what you expect them to be based on your `CREATE TABLE` statement, note that MySQL sometimes changes data types when you create or alter a table. The conditions for which this occurs are described in [Section 13.1.5.1, “Silent Column Specification Changes”](#).

The `FULL` keyword causes the output to include the privileges you have as well as any per-column comments for each column.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. In other words, these two statements are equivalent:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

SHOW FIELDS is a synonym for SHOW COLUMNS. You can also list a table's columns with the **mysqlshow *db_name tbl_name*** command.

The DESCRIBE statement provides information similar to SHOW COLUMNS. See [Section 13.3.1, “DESCRIBE Syntax”](#).

13.5.4.4. SHOW CREATE DATABASE Syntax

```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

Shows the CREATE DATABASE statement that creates the given database. SHOW CREATE SCHEMA is a synonym for SHOW CREATE DATABASE as of MySQL 5.0.2.

```
mysql> SHOW CREATE DATABASE test\G
***** 1. row *****
      Database: test
Create Database: CREATE DATABASE `test`
                  /*!40100 DEFAULT CHARACTER SET latin1 */
```

```
mysql> SHOW CREATE SCHEMA test\G
***** 1. row *****
      Database: test
Create Database: CREATE DATABASE `test`
                  /*!40100 DEFAULT CHARACTER SET latin1 */
```

SHOW CREATE DATABASE quotes table and column names according to the value of the SQL_QUOTE_SHOW_CREATE option. See [Section 13.5.3, “SET Syntax”](#).

13.5.4.5. SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION Syntax

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

This statement is a MySQL extension. Similar to SHOW CREATE TABLE, it returns the exact string that can be used to re-create the named routine.

```
mysql> SHOW CREATE FUNCTION test.hello\G
***** 1. row *****
      Function: hello
      sql_mode:
Create Function: CREATE FUNCTION `test`.`hello` (s CHAR(20)) RETURNS
```

```
RETURN CONCAT('Hello, ',s, '!')
```

13.5.4.6. SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows the CREATE TABLE statement that creates the given table. As of MySQL 5.0.1, this statement also works with views.

```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
      Table: t
Create Table: CREATE TABLE t (
  id INT(11) default NULL auto_increment,
  s char(60) default NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM
```

SHOW CREATE TABLE quotes table and column names according to the value of the SQL_QUOTE_SHOW_CREATE option. See [Section 13.5.3, “SET Syntax”](#).

13.5.4.7. SHOW CREATE VIEW Syntax

```
SHOW CREATE VIEW view_name
```

This statement shows a CREATE VIEW statement that creates the given view.

```
mysql> SHOW CREATE VIEW v;
+-----+-----+-----+-----+-----+-----+
| View | Create View |
+-----+-----+-----+-----+
| v    | CREATE VIEW `test`.`v` AS select 1 AS `a`,2 AS `b` |
+-----+-----+-----+-----+-----+-----+-----+
```

This statement was added in MySQL 5.0.1.

Prior to MySQL 5.0.11, the output columns from this statement were shown as Table and Create Table.

Use of SHOW CREATE VIEW requires the SHOW VIEW privilege and the SELECT privilege for the view in question.

You can also obtain information about view objects from INFORMATION_SCHEMA, which contains a VIEWS table. See [Section 20.15, “The INFORMATION_SCHEMA VIEWS Table”](#).

13.5.4.8. SHOW DATABASES Syntax

```
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']
```

SHOW DATABASES lists the databases on the MySQL server host. SHOW SCHEMAS is a synonym for SHOW DATABASES as of MySQL 5.0.2.

You see only those databases for which you have some kind of privilege, unless you have the global SHOW DATABASES privilege. You can also get this list using the **mysqlshow** command.

If the server was started with the `--skip-show-database` option, you cannot use this statement at all unless you have the SHOW DATABASES privilege.

13.5.4.9. SHOW ENGINE Syntax

```
SHOW ENGINE engine_name {LOGS | STATUS }
```

SHOW ENGINE displays log or status information about a storage engine. The following statements currently are supported:

```
SHOW ENGINE BDB LOGS  
SHOW ENGINE INNODB STATUS
```

SHOW ENGINE BDB LOGS displays status information about existing BDB log files. It returns the following fields:

- File
The full path to the log file.
- Type
The log file type (BDB for Berkeley DB log files).
- Status

The status of the log file (FREE if the file can be removed, or IN USE if the file is needed by the transaction subsystem)

SHOW ENGINE INNODB STATUS displays extensive information about the state of the InnoDB storage engine.

The InnoDB Monitors provide additional information about InnoDB processing. See [Section 14.2.11.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”](#).

Older (and now deprecated) synonyms for these statements are SHOW [BDB] LOGS and SHOW INNODB STATUS.

13.5.4.10. SHOW ENGINES Syntax

SHOW [STORAGE] ENGINES

SHOW ENGINES displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is. SHOW TABLE TYPES is a deprecated synonym.

```
mysql> SHOW ENGINES\G
***** 1. row *****
Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
***** 2. row *****
Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
***** 3. row *****
Engine: HEAP
Support: YES
Comment: Alias for MEMORY
***** 4. row *****
Engine: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
***** 5. row *****
Engine: MRG_MYISAM
Support: YES
Comment: Alias for MERGE
***** 6. row *****
Engine: ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MyISAM
```

```

***** 7. row *****
Engine: MRG_ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MERGE
***** 8. row *****
Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
***** 9. row *****
Engine: INNODB
Support: YES
Comment: Alias for INNODB
***** 10. row *****
Engine: BDB
Support: YES
Comment: Supports transactions and page-level locking
***** 11. row *****
Engine: BERKELEYDB
Support: YES
Comment: Alias for BDB
***** 12. row *****
Engine: NDBCLUSTER
Support: NO
Comment: Clustered, fault-tolerant, memory-based tables
***** 13. row *****
Engine: NDB
Support: NO
Comment: Alias for NDBCLUSTER
***** 14. row *****
Engine: EXAMPLE
Support: NO
Comment: Example storage engine
***** 15. row *****
Engine: ARCHIVE
Support: YES
Comment: Archive storage engine
***** 16. row *****
Engine: CSV
Support: NO
Comment: CSV storage engine
***** 17. row *****
Engine: FEDERATED
Support: YES
Comment: Federated MySQL storage engine
***** 18. row *****
Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write to it disappea

```

The Support value indicates whether the particular storage engine is supported, and which is the default engine. For example, if the server is started with the `--default-table-type=InnoDB` option, the Support value for the InnoDB row has the value DEFAULT. See [Chapter 14, Storage Engines and Table Types](#).

13.5.4.11. SHOW ERRORS Syntax

```
SHOW ERRORS [LIMIT [offset,] row_count]  
SHOW COUNT(*) ERRORS
```

This statement is similar to SHOW WARNINGS, except that instead of displaying errors, warnings, and notes, it displays only errors.

The LIMIT clause has the same syntax as for the SELECT statement. See [Section 13.2.7, “SELECT Syntax”](#).

The SHOW COUNT(*) ERRORS statement displays the number of errors. You can also retrieve this number from the error_count variable:

```
SHOW COUNT(*) ERRORS;  
SELECT @@error_count;
```

For more information, see [Section 13.5.4.25, “SHOW WARNINGS Syntax”](#).

13.5.4.12. SHOW GRANTS Syntax

```
SHOW GRANTS FOR user
```

This statement lists the GRANT statement or statements that must be issued to duplicate the privileges that are granted to a MySQL user account. The account is named using the same format as for the GRANT statement; for example, 'jeffrey'@'localhost'. The user and host parts of the account name correspond to the User and Host column values of the user table row for the account.

```
mysql> SHOW GRANTS FOR 'root'@'localhost';  
+-----  
| Grants for root@localhost  
+-----  
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTIO  
+-----
```

To list the privileges granted to the account that you are using to connect to the server, you can use any of the following statements:

```
SHOW GRANTS;  
SHOW GRANTS FOR CURRENT_USER;  
SHOW GRANTS FOR CURRENT_USER();
```

As of MySQL 5.0.24, if `SHOW GRANTS FOR CURRENT_USER` (or any of the equivalent syntaxes) is used in `DEFINER` context, such as within a stored procedure that is defined with `SQL SECURITY DEFINER`, the grants displayed are those of the definer and not the invoker.

`SHOW GRANTS` displays only the privileges granted explicitly to the named account. Other privileges might be available to the account, but they are not displayed. For example, if an anonymous account exists, the named account might be able to use its privileges, but `SHOW GRANTS` will not display them.

13.5.4.13. SHOW INDEX Syntax

```
SHOW INDEX FROM tbl_name [FROM db_name]
```

`SHOW INDEX` returns table index information. The format resembles that of the `SQLStatistics` call in ODBC.

`SHOW INDEX` returns the following fields:

- `Table`

The name of the table.

- `Non_unique`

0 if the index cannot contain duplicates, 1 if it can.

- `Key_name`

The name of the index.

- `Seq_in_index`

The column sequence number in the index, starting with 1.

- Column_name

The column name.

- Collation

How the column is sorted in the index. In MySQL, this can have values 'A' (Ascending) or NULL (Not sorted).

- Cardinality

An estimate of the number of unique values in the index. This is updated by running `ANALYZE TABLE` or `myisamchk -a`. Cardinality is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- Sub_part

The number of indexed characters if the column is only partly indexed, NULL if the entire column is indexed.

- Packed

Indicates how the key is packed. NULL if it is not.

- Null

Contains YES if the column may contain NULL. If not, the column contains NO as of MySQL 5.0.3, and ' ' before that.

- Index_type

The index method used (BTREE, FULLTEXT, HASH, RTREE).

- Comment

Various remarks.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;  
SHOW INDEX FROM mydb.mytable;
```

SHOW KEYS is a synonym for SHOW INDEX. You can also list a table's indexes with the **mysqlshow -k db_name tbl_name** command.

13.5.4.14. SHOW INNODB STATUS Syntax

```
SHOW INNODB STATUS
```

In MySQL 5.0, this is a deprecated synonym for SHOW ENGINE INNODB STATUS. See [Section 13.5.4.9, “SHOW ENGINE Syntax”](#).

13.5.4.15. SHOW LOGS Syntax

```
SHOW [BDB] LOGS
```

In MySQL 5.0, this is a deprecated synonym for SHOW ENGINE BDB LOGS. See [Section 13.5.4.9, “SHOW ENGINE Syntax”](#).

13.5.4.16. SHOW OPEN TABLES Syntax

```
SHOW OPEN TABLES [FROM db_name] [LIKE 'pattern']
```

SHOW OPEN TABLES lists the non-TEMPORARY tables that are currently open in the table cache. See [Section 7.4.8, “How MySQL Opens and Closes Tables”](#).

SHOW OPEN TABLES returns the following fields:

- Database

The database containing the table.

- Table

The table name.

- In_use

The number of times the table currently is in use by queries. If the count is

zero, the table is open, but not currently being used.

- Name_locked

Whether the table name is locked. Name locking is used for operations such as dropping or renaming tables.

The FROM and LIKE clauses may be used as of MySQL 5.0.12.

13.5.4.17. SHOW PRIVILEGES Syntax

SHOW PRIVILEGES

SHOW PRIVILEGES shows the list of system privileges that the MySQL server supports. The exact list of privileges depends on the version of your server.

```
mysql> SHOW PRIVILEGES\G
***** 1. row *****
Privilege: Alter
Context: Tables
Comment: To alter the table
***** 2. row *****
Privilege: Alter routine
Context: Functions,Procedures
Comment: To alter or drop stored functions/procedures
***** 3. row *****
Privilege: Create
Context: Databases,Tables,Indexes
Comment: To create new databases and tables
***** 4. row *****
Privilege: Create routine
Context: Functions,Procedures
Comment: To use CREATE FUNCTION/PROCEDURE
***** 5. row *****
Privilege: Create temporary tables
Context: Databases
Comment: To use CREATE TEMPORARY TABLE
...
```

13.5.4.18. SHOW PROCEDURE STATUS and SHOW FUNCTION STATUS Syntax

SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']

This statement is a MySQL extension. It returns characteristics of routines, such

as the database, name, type, creator, and creation and modification dates. If no pattern is specified, the information for all stored procedures or all stored functions is listed, depending on which statement you use.

```
mysql> SHOW FUNCTION STATUS LIKE 'hello'\G
***** 1. row *****
      Db: test
      Name: hello
      Type: FUNCTION
      Definer: testuser@localhost
      Modified: 2004-08-03 15:29:37
      Created: 2004-08-03 15:29:37
      Security_type: DEFINER
      Comment:
```

You can also get information about stored routines from the ROUTINES table in INFORMATION_SCHEMA. See [Section 20.14, “The INFORMATION_SCHEMA ROUTINES Table”](#).

13.5.4.19. SHOW PROCESSLIST Syntax

```
SHOW [FULL] PROCESSLIST
```

SHOW PROCESSLIST shows you which threads are running. You can also get this information using the **mysqladmin processlist** command. If you have the PROCESS privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). See [Section 13.5.5.3, “KILL Syntax”](#). If you do not use the FULL keyword, only the first 100 characters of each statement are shown in the Info field.

This statement is very useful if you get the “too many connections” error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the SUPER privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

The output of SHOW PROCESSLIST may look like this:

```
mysql> SHOW FULL PROCESSLIST\G
***** 1. row *****
Id: 1
User: system user
Host:
```

```

db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
***** 2. row *****
Id: 2
User: system user
Host:
db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave I/O thread to u
Info: NULL
***** 3. row *****
Id: 3112
User: replikator
Host: artemis:2204
db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be update
Info: NULL
***** 4. row *****
Id: 3113
User: replikator
Host: iconnect2:45781
db: NULL
Command: Binlog Dump
Time: 2086
State: Has sent all binlog to slave; waiting for binlog to be update
Info: NULL
***** 5. row *****
Id: 3123
User: stefan
Host: localhost
db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
5 rows in set (0.00 sec)

```

The columns have the following meaning:

- Id

The connection identifier.

- User

The MySQL user who issued the statement. If this is `system` user, it refers to a non-client thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. For `system` user, there is no host specified in the `Host` column.

- Host

The hostname of the client issuing the statement (except for `system` user where there is no host). `SHOW PROCESSLIST` reports the hostname for TCP/IP connections in `host_name:client_port` format to make it easier to determine which client is doing what.

- db

The default database, if one is selected, otherwise `NULL`.

- Command

The value of that column corresponds to the `COM_xxx` commands of the client/server protocol. See [Section 5.2.4, “Server Status Variables”](#)

The `Command` value may be one of the following: `Binlog Dump`, `Change user`, `Close stmt`, `Connect`, `Connect Out`, `Create DB`, `Daemon`, `Debug`, `Delayed insert`, `Drop DB`, `Error`, `Execute`, `Fetch`, `Field List`, `Init DB`, `Kill`, `Long Data`, `Ping`, `Prepare`, `Processlist`, `Query`, `Quit`, `Refresh`, `Register Slave`, `Reset stmt`, `Set option`, `Shutdown`, `Sleep`, `Statistics`, `Table Dump`, `Time`

- Time

The time in seconds between the start of the statement or command and `now`.

- State

An action, event, or state, which can be one of the following: `After create`, `Analyzing`, `Changing master`, `Checking master version`, `Checking table`, `Connecting to master`, `Copying to group table`,

Copying to tmp table, Creating delayed handler, Creating index, Creating sort index, Creating table from master dump, Creating tmp table, Execution of init_command, FULLTEXT initialization, Finished reading one binlog; switching to next binlog, Flushing tables, Killed, Killing slave, Locked, Making temp file , Opening master dump table, Opening table, Opening tables, Processing request, Purging old relay logs, Queueing master event to the relay log, Reading event from the relay log, Reading from net, Reading master dump table data, Rebuilding the index on master dump table, Reconnecting after a failed binlog dump request, Reconnecting after a failed master event read, Registering slave on master, Removing duplicates, Reopen tables, Repair by sorting, Repair done, Repair with keycache, Requesting binlog dump, Rolling back, Saving state, Searching rows for update, Sending binlog event to slave, Sending data, Sorting for group, Sorting for order, Sorting index, Sorting result, System lock, Table lock, Thread initialized, Updating, User lock, Waiting for INSERT, Waiting for master to send event, Waiting for master update, Waiting for slave mutex on exit, Waiting for table, Waiting for tables, Waiting for the next event in relay log, Waiting on cond, Waiting to finalize termination, Waiting to reconnect after a failed binlog dump request, Waiting to reconnect after a failed master event read, Writing to net, allocating local table, cleaning up, closing tables, converting HEAP to MyISAM, copy to tmp table, creating table, deleting from main table, deleting from reference tables, discard_or_import_tablespace, end, freeing items, got handler lock, got old table, info, init, insert, logging slow query, login, preparing, purging old relay logs, query end, removing tmp table, rename, rename result table, reschedule, setup, starting slave, statistics, storing row into queue, unauthenticated user, update, updating, updating main table, updating reference tables, upgrading lock, waiting for delay_list, waiting for handler insert, waiting for handler lock, waiting for handler open, Waiting for event from ndbcluster

The most common State values are described in the rest of this section. Most of the other State values are useful only for finding bugs in the server. See also [Section 6.3, “Replication Implementation Details”](#), for additional information about process states for replication servers.

For the `SHOW PROCESSLIST` statement, the value of State is `NULL`.

- Info

The statement that the thread is executing, or `NULL` if it is not executing any statement.

Some State values commonly seen in the output from `SHOW PROCESSLIST`:

- Checking table

The thread is performing a table check operation.

- Closing tables

Means that the thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, you should verify that you do not have a full disk and that the disk is not in very heavy use.

- Connect Out

A replication slave is connecting to its master.

- Copying to group table

If a statement has different `ORDER BY` and `GROUP BY` criteria, the rows are sorted by group and copied to a temporary table.

- Copying to tmp table

The server is copying to a temporary table in memory.

- Copying to tmp table on disk

The server is copying to a temporary table on disk. The temporary result set was larger than `tmp_table_size` and the thread is changing the temporary table from in-memory to disk-based format to save memory.

- Creating tmp table

The thread is creating a temporary table to hold a part of the result for the

query.

- deleting from main table

The server is executing the first part of a multiple-table delete. It is deleting only from the first table, and saving fields and offsets to be used for deleting from the other (reference) tables.

- deleting from reference tables

The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- Flushing tables

The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables.

- FULLTEXT initialization

The server is preparing to perform a natural-language full-text search.

- Killed

Someone has sent a `KILL` statement to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- Locked

The query is locked by another query.

- Sending data

The thread is processing rows for a `SELECT` statement and also is sending data to the client.

- Sorting for group

The thread is doing a sort to satisfy a GROUP BY.

- Sorting for order

The thread is doing a sort to satisfy a ORDER BY.

- Opening tables

The thread is trying to open a table. This should be a very fast procedure, unless something prevents opening. For example, an ALTER TABLE or a LOCK TABLE statement can prevent opening a table until the statement is finished.

- Reading from net

The server is reading a packet from the network.

- Removing duplicates

The query was using SELECT DISTINCT in such a way that MySQL could not optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- Reopen table

The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- Repair by sorting

The repair code is using a sort to create indexes.

- Repair with keycache

The repair code is using creating keys one by one through the key cache. This is much slower than Repair by sorting.

- Searching rows for update

The thread is doing a first phase to find all matching rows before updating

them. This has to be done if the UPDATE is changing the index that is used to find the involved rows.

- Sleeping

The thread is waiting for the client to send a new statement to it.

- statistics

The server is calculating statistics to develop a query execution plan.

- System lock

The thread is waiting to get an external system lock for the table. If you are not using multiple **mysqld** servers that are accessing the same tables, you can disable system locks with the `--skip-external-locking` option.

- unauthenticated user

The state of a thread that has become associated with a client connection but for which authentication of the client user has not yet been done.

- Upgrading lock

The INSERT DELAYED handler is trying to get a lock for the table to insert rows.

- Updating

The thread is searching for rows to update and is updating them.

- updating main table

The server is executing the first part of a multiple-table update. It is updating only the first table, and saving fields and offsets to be used for updating the other (reference) tables.

- updating reference tables

The server is executing the second part of a multiple-table update and updating the matched rows from the other tables.

- User Lock

The thread is waiting on a `GET_LOCK()`.

- Waiting for event from `ndbcluster`

The server is acting as an SQL node in a MySQL Cluster, and is connected to a cluster management node.

- Waiting for tables

The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

This notification takes place if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES tbl_name`, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

- waiting for handler insert

The `INSERT DELAYED` handler has processed all pending inserts and is waiting for new ones.

- Writing to net

The server is writing a packet to the network.

Most states correspond to very quick operations. If a thread stays in any of these states for many seconds, there might be a problem that needs to be investigated.

13.5.4.20. `SHOW STATUS` Syntax

```
SHOW [GLOBAL | SESSION] STATUS [LIKE 'pattern']
```

`SHOW STATUS` provides server status information. This information also can be obtained using the **`mysqladmin extended-status`** command.

Partial output is shown here. The list of names and values may be different for your server. The meaning of each variable is given in [Section 5.2.4, “Server Status Variables”](#).

```
mysql> SHOW STATUS;
+-----+-----+
| Variable_name      | Value      |
+-----+-----+
| Aborted_clients    | 0          |
| Aborted_connects   | 0          |
| Bytes_received     | 155372598 |
| Bytes_sent         | 1176560426|
| Connections        | 30023      |
| Created_tmp_disk_tables | 0          |
| Created_tmp_tables | 8340       |
| Created_tmp_files  | 60         |
| ..                | ..         |
| Open_tables        | 1          |
| Open_files         | 2          |
| Open_streams       | 0          |
| Opened_tables      | 44600      |
| Questions          | 2026873    |
| ..                | ..         |
| Table_locks_immediate | 1920382    |
| Table_locks_waited  | 0          |
| Threads_cached     | 0          |
| Threads_created    | 30022      |
| Threads_connected  | 1          |
| Threads_running    | 1          |
| Uptime             | 80380      |
+-----+-----+
```

With a LIKE clause, the statement displays only rows for those variables with names that match the pattern:

```
mysql> SHOW STATUS LIKE 'Key%';
+-----+-----+
| Variable_name      | Value      |
+-----+-----+
| Key_blocks_used    | 14955      |
| Key_read_requests  | 96854827   |
| Key_reads          | 162040     |
| Key_write_requests | 7589728    |
| Key_writes         | 3813196    |
+-----+-----+
```

The GLOBAL and SESSION options are new in MySQL 5.0.2. With the GLOBAL

modifier, `SHOW STATUS` displays the status values for all connections to MySQL. With `SESSION`, it displays the status values for the current connection. If no modifier is present, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`.

Some status variables have only a global value. For these, you get the same value for both `GLOBAL` and `SESSION`.

Note: Before MySQL 5.0.2, `SHOW STATUS` returned global status values. Because the default as of 5.0.2 is to return session values, this is incompatible with previous versions. To issue a `SHOW STATUS` statement that will retrieve global status values for all versions of MySQL, write it like this:

```
SHOW /*!50002 GLOBAL */ STATUS;
```

13.5.4.21. `SHOW TABLE STATUS` Syntax

```
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
```

`SHOW TABLE STATUS` works like `SHOW TABLE`, but provides a lot of information about each table. You can also get this list using the `mysqlshow --status db_name` command.

As of MySQL 5.0.1, this statement also displays information about views.

`SHOW TABLE STATUS` returns the following fields:

- Name

The name of the table.

- Engine

The storage engine for the table. See [Chapter 14, Storage Engines and Table Types](#).

- Version

The version number of the table's `.frm` file.

- Row_format

The row storage format (Fixed, Dynamic, Compressed, Redundant, Compact). Starting with MySQL/InnoDB 5.0.3, the format of InnoDB tables is reported as Redundant or Compact. Prior to 5.0.3, InnoDB tables are always in the Redundant format.

- Rows

The number of rows. Some storage engines, such as MyISAM, store the exact count. For other storage engines, such as InnoDB, this value is an approximation, and may vary from the actual value by as much as 40 to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

The Rows value is NULL for tables in the INFORMATION_SCHEMA database.

- Avg_row_length

The average row length.

- Data_length

The length of the data file.

- Max_data_length

The maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

- Index_length

The length of the index file.

- Data_free

The number of allocated but unused bytes.

- Auto_increment

The next AUTO_INCREMENT value.

- Create_time

When the table was created.

- `Update_time`

When the data file was last updated. For some storage engines, this value is `NULL`. For example, InnoDB stores multiple tables in its tablespace and the data file timestamp does not apply.

- `Check_time`

When the table was last checked. Not all storage engines update this time, in which case the value is always `NULL`.

- `Collation`

The table's character set and collation.

- `Checksum`

The live checksum value (if any).

- `Create_options`

Extra options used with `CREATE TABLE`.

- `Comment`

The comment used when creating the table (or information as to why MySQL could not access the table information).

In the table comment, InnoDB tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table.

For MEMORY tables, the `Data_length`, `Max_data_length`, and `Index_length` values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

Beginning with MySQL 5.0.3, for NDB cluster tables, the output of this

statement shows appropriate values for the Avg_row_length and Data_length columns, with the exception that BLOB columns are not taken into account. In addition, the number of replicas is now shown in the Comment column (as number_of_replicas).

For views, all the fields displayed by SHOW TABLE STATUS are NULL except that Name indicates the view name and Comment says view.

13.5.4.22. SHOW TABLES Syntax

```
SHOW [FULL] TABLES [FROM db_name] [LIKE 'pattern']
```

SHOW TABLES lists the non-TEMPORARY tables in a given database. You can also get this list using the **mysqlshow db_name** command.

Before MySQL 5.0.1, the output from SHOW TABLES contains a single column of table names. Beginning with MySQL 5.0.1, this statement also lists any views in the database. As of MySQL 5.0.2, the FULL modifier is supported such that SHOW FULL TABLES displays a second output column. Values for the second column are BASE TABLE for a table and VIEW for a view.

Note: If you have no privileges for a table, the table does not show up in the output from SHOW TABLES or **mysqlshow db_name**.

13.5.4.23. SHOW TRIGGERS Syntax

```
SHOW TRIGGERS [FROM db_name] [LIKE expr]
```

SHOW TRIGGERS lists the triggers currently defined on the MySQL server. This statement requires the SUPER privilege. It was implemented in MySQL 5.0.10.

For the trigger ins_sum as defined in [Section 18.3, “Using Triggers”](#), the output of this statement is as shown here:

```
mysql> SHOW TRIGGERS LIKE 'acc%\G
***** 1. row *****
  Trigger: ins_sum
    Event: INSERT
    Table: account
Statement: SET @sum = @sum + NEW.amount
    Timing: BEFORE
```

```
Created: NULL
sql_mode:
Definer: myname@localhost
```

Note: When using a LIKE clause with SHOW TRIGGERS, the expression to be matched (*expr*) is compared with the name of the table on which the trigger is declared, and not with the name of the trigger:

```
mysql> SHOW TRIGGERS LIKE 'ins%';
Empty set (0.01 sec)
```

A brief explanation of the columns in the output of this statement is shown here:

- Trigger

The name of the trigger.

- Event

The event that causes trigger activation: one of 'INSERT', 'UPDATE', or 'DELETE'.

- Table

The table for which the trigger is defined.

- Statement

The statement to be executed when the trigger is activated. This is the same as the text shown in the ACTION_STATEMENT column of INFORMATION_SCHEMA.TRIGGERS.

- Timing

One of the two values 'BEFORE' or 'AFTER'.

- Created

Currently, the value of this column is always NULL.

- sql_mode

The SQL mode in effect when the trigger executes. This column was added in MySQL 5.0.11.

- Definer

The account that created the trigger. This column was added in MySQL 5.0.17.

You must have the SUPER privilege to execute SHOW TRIGGERS.

See also [Section 20.16, “The INFORMATION_SCHEMA TRIGGERS Table”](#).

13.5.4.24. SHOW VARIABLES Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
```

SHOW VARIABLES shows the values of MySQL system variables. This information also can be obtained using the **mysqladmin variables** command.

With the GLOBAL modifier, SHOW VARIABLES displays the values that are used for new connections to MySQL. With SESSION, it displays the values that are in effect for the current connection. If no modifier is present, the default is SESSION. LOCAL is a synonym for SESSION.

If the default system variable values are unsuitable, you can set them using command options when **mysqld** starts, and most can be changed at runtime with the SET statement. See [Section 5.2.3, “Using System Variables”](#), and [Section 13.5.3, “SET Syntax”](#).

Partial output is shown here. The list of names and values may be different for your server. [Section 5.2.2, “Server System Variables”](#), describes the meaning of each variable, and [Section 7.5.2, “Tuning Server Parameters”](#), provides information about tuning them.

```
mysql> SHOW VARIABLES;
```

Variable_name	Value
auto_increment_increment	1
auto_increment_offset	1
automatic_sp_privileges	ON

back_log	50
basedir	/
bdb_cache_size	8388600
bdb_home	/var/lib/mysql/
bdb_log_buffer_size	32768
...	
max_connections	100
max_connect_errors	10
max_delayed_threads	20
max_error_count	64
max_heap_table_size	16777216
max_join_size	4294967295
max_relay_log_size	0
max_sort_length	1024
...	
time_zone	SYSTEM
timed_mutexes	OFF
tmp_table_size	33554432
tmpdir	
transaction_alloc_block_size	8192
transaction_prealloc_size	4096
tx_isolation	REPEATABLE-READ
updatable_views_with_limit	YES
version	5.0.19-Max
version_comment	MySQL Community Edition - Max (G
version_compile_machine	i686
version_compile_os	pc-linux-gnu
wait_timeout	28800

With a LIKE clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a LIKE clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the ‘%’ wildcard character in a LIKE clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because ‘_’ is a wildcard that matches any single character, you should escape it as ‘_’ to match it literally. In practice, this is rarely

necessary.

13.5.4.25. SHOW WARNINGS Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]  
SHOW COUNT(*) WARNINGS
```

SHOW WARNINGS shows the error, warning, and note messages that resulted from the last statement that generated messages, or nothing if the last statement that used a table generated no messages. A related statement, SHOW ERRORS, shows only the errors. See [Section 13.5.4.11, “SHOW ERRORS Syntax”](#).

The list of messages is reset for each new statement that uses a table.

The SHOW COUNT(*) WARNINGS statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the warning_count variable:

```
SHOW COUNT(*) WARNINGS;  
SELECT @@warning_count;
```

The value of warning_count might be greater than the number of messages displayed by SHOW WARNINGS if the max_error_count system variable is set so low that not all messages are stored. An example shown later in this section demonstrates how this can happen.

The LIMIT clause has the same syntax as for the SELECT statement. See [Section 13.2.7, “SELECT Syntax”](#).

The MySQL server sends back the total number of errors, warnings, and notes resulting from the last statement. If you are using the C API, this value can be obtained by calling mysql_warning_count(). See [Section 22.2.3.71, “mysql_warning_count\(\)”](#).

Warnings are generated for statements such as LOAD DATA INFILE and DML statements such as INSERT, UPDATE, CREATE TABLE, and ALTER TABLE.

The following DROP TABLE statement results in a note:

```
mysql> DROP TABLE IF EXISTS no_such_table;  
mysql> SHOW WARNINGS;
```

```

+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note | 1051 | Unknown table 'no_such_table' |
+-----+-----+-----+

```

Here is a simple example that shows a syntax warning for CREATE TABLE and conversion warnings for INSERT:

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4)) TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
'ENGINE=storage_engine' instead
1 row in set (0.00 sec)
```

```
mysql> INSERT INTO t1 VALUES(10, 'mysql'), (NULL, 'test'),
-> (300, 'Open Source');
```

```
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 4
```

```
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 1
***** 2. row *****
```

```
Level: Warning
Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row
***** 3. row *****
```

```
Level: Warning
Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
***** 4. row *****
```

```
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)
```

The maximum number of error, warning, and note messages to store is controlled by the `max_error_count` system variable. By default, its value is 64. To change the number of messages you want stored, change the value of `max_error_count`. In the following example, the ALTER TABLE statement

produces three warning messages, but only one is stored because `max_error_count` has been set to 1:

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_error_count | 64    |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SET max_error_count=1;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 3
```

```
mysql> SELECT @@warning_count;
```

```
+-----+
| @@warning_count |
+-----+
|                3 |
+-----+
1 row in set (0.01 sec)
```

```
mysql> SHOW WARNINGS;
```

```
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

To disable warnings, set `max_error_count` to 0. In this case, `warning_count` still indicates how many warnings have occurred, but none of the messages are stored.

As of MySQL 5.0.3, you can set the `SQL_NOTES` session variable to 0 to cause Note-level warnings not to be recorded.

13.5.5. Other Administrative Statements

13.5.5.1. CACHE INDEX Syntax

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  IN key_cache_name
```

```
tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

The `CACHE INDEX` statement assigns table indexes to a specific key cache. It is used only for MyISAM tables.

The following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+-----+-----+-----+-----+
| Table  | Op                | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+-----+-----+-----+-----+
```

The syntax of `CACHE INDEX` enables you to specify that only particular indexes from a table should be assigned to the cache. The current implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters can be accessed as members of a structured system variable. See [Section 5.2.3.1, “Structured System Variables”](#).

A key cache must exist before you can assign indexes to it:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it become assigned to the default key cache again.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

13.5.5.2. FLUSH Syntax

```
FLUSH [LOCAL | NO_WRITE_TO_BINLOG] flush_option [, flush_option] ...
```

The FLUSH statement clears or reloads various internal caches used by MySQL. To execute FLUSH, you must have the RELOAD privilege.

The RESET statement is similar to FLUSH. See [Section 13.5.5.5, “RESET Syntax”](#).

flush_option can be any of the following:

- HOSTS

Empties the host cache tables. You should flush the host tables if some of your hosts change IP number or if you get the error message Host 'host_name' is blocked. When more than `max_connect_errors` errors occur successively for a given host while connecting to the MySQL server, MySQL assumes that something is wrong and blocks the host from further connection requests. Flushing the host tables allows the host to attempt to connect again. See [Section A.2.5, “Host 'host_name' is blocked”](#). You can start **mysqld** with `--max_connect_errors=999999999` to avoid this error message.

- DES_KEY_FILE

Reloads the DES keys from the file that was specified with the `--des-key-file` option at server startup time.

- LOGS

Closes and reopens all log files. If binary logging is enabled, the sequence number of the binary log file is incremented by one relative to the previous file. On Unix, this is the same thing as sending a SIGHUP signal to the **mysqld** server (except on some Mac OS X 10.3 versions where **mysqld** ignores SIGHUP and SIGQUIT).

If the server was started with the `--log-error` option, `FLUSH LOGS` causes the error log is renamed with a suffix of `-old` and **mysqld** creates a new empty log file. No renaming occurs if the `--log-error` option was not given.

- `MASTER` (*DEPRECATED*). Deletes all binary logs, resets the binary log index file and creates a new binary log. Deprecated in favor of `RESET MASTER`, supported for backwards compatibility only See [Section 13.6.1.2, “RESET MASTER Syntax”](#).

- `PRIVILEGES`

Reloads the privileges from the grant tables in the `mysql` database.

- `QUERY CACHE`

Defragment the query cache to better utilize its memory. `FLUSH QUERY CACHE` does not remove any queries from the cache, unlike `RESET QUERY CACHE`.

- `SLAVE` (*DEPRECATED*). Resets all replication slave parameters, including relay log files and replication position in the master's binary logs. Deprecated in favor of `RESET SLAVE`, supported for backwards compatibility only. See [Section 13.6.2.5, “RESET SLAVE Syntax”](#).

- `STATUS`

Resets most status variables to zero. This is something you should use only when debugging a query. See [Section 1.8, “How to Report Bugs or Problems”](#).

- `{TABLE | TABLES} [tbl_name [, tbl_name] ...]`

When no tables are named, closes all open tables and forces all tables in use to be closed. This also flushes the query cache. With one or more table names, flushes only the given tables. `FLUSH TABLES` also removes all query results from the query cache, like the `RESET QUERY CACHE` statement.

- `TABLES WITH READ LOCK`

Closes all open tables and locks all tables for all databases with a read lock until you execute `UNLOCK TABLES`. This is very convenient way to get backups if you have a filesystem such as Veritas that can take snapshots in time.

- `USER_RESOURCES`

Resets all per-hour user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. `FLUSH USER_RESOURCES` does not apply to the limit on maximum simultaneous connections. See [Section 13.5.1.3, “GRANT Syntax”](#).

`FLUSH` statements are written to the binary log unless the optional `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`) is used. This is done so that `FLUSH` statements used on a MySQL server acting as a replication master will be replicated by default to the replication slave.

Note: `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK` are not logged in any case because they would cause problems if replicated to a slave.

You can also access some of these statements with the **mysqladmin** utility, using the `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, or `flush-tables` commands.

Using `FLUSH` statements within stored functions or triggers is not supported in MySQL 5.0. However, you may use `FLUSH` in stored procedures, so long as these are not called from stored functions or triggers. See [Section I.1, “Restrictions on Stored Routines and Triggers”](#).

See also [Section 13.5.5.5, “RESET Syntax”](#), for information about how the `RESET` statement is used with replication.

13.5.5.3. `KILL` Syntax

```
KILL [CONNECTION | QUERY] thread_id
```

Each connection to **mysqld** runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` statement and kill a thread with the `KILL thread_id` statement.

In MySQL 5.0.0, KILL allows the optional CONNECTION or QUERY modifier:

- KILL CONNECTION is the same as KILL with no modifier: It terminates the connection associated with the given *thread_id*.
- KILL QUERY terminates the statement that the connection is currently executing, but leaves the connection itself intact.

If you have the PROCESS privilege, you can see all threads. If you have the SUPER privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

You can also use the **mysqladmin processlist** and **mysqladmin kill** commands to examine and kill threads.

Note: You cannot use KILL with the Embedded MySQL Server library, because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

When you use KILL, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die, because the kill flag is checked only at specific intervals:

- In SELECT, ORDER BY and GROUP BY loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.
- During ALTER TABLE, the kill flag is checked before each block of rows are read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.
- During UPDATE or DELETE operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. Note that if you are not using transactions, the changes are not rolled back.
- GET_LOCK() aborts and returns NULL.
- An INSERT DELAYED thread quickly flushes (inserts) all rows it has in memory and then terminates.

- If the thread is in the table lock handler (state: Locked), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.
- **Warning:** Killing a REPAIR TABLE or OPTIMIZE TABLE operation on a MyISAM table results in a table that is corrupted and unusable. Any reads or writes to such a table fail until you optimize or repair it again (without interruption).

13.5.5.4. LOAD INDEX INTO CACHE Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...
```

```
tbl_index_list:
  tbl_name
  [[INDEX|KEY] (index_name[, index_name] ...)]
  [IGNORE LEAVES]
```

The LOAD INDEX INTO CACHE statement preloads a table index into the key cache to which it has been assigned by an explicit CACHE INDEX statement, or into the default key cache otherwise. LOAD INDEX INTO CACHE is used only for MyISAM tables.

The IGNORE LEAVES modifier causes only blocks for the non-leaf nodes of the index to be preloaded.

The following statement preloads nodes (index blocks) of indexes for the tables t1 and t2:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+-----+-----+-----+-----+
| Table  | Op           | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+-----+-----+-----+-----+
```

This statement preloads all index blocks from t1. It preloads only blocks for the non-leaf nodes from t2.

The syntax of `LOAD INDEX INTO CACHE` enables you to specify that only particular indexes from a table should be preloaded. The current implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

13.5.5.5. RESET Syntax

```
RESET reset_option [, reset_option] ...
```

The `RESET` statement is used to clear the state of various server operations. You must have the `RELOAD` privilege to execute `RESET`.

`RESET` acts as a stronger version of the `FLUSH` statement. See [Section 13.5.5.2, “FLUSH Syntax”](#).

reset_option can be any of the following:

- MASTER

Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file. (Known as `FLUSH MASTER` in versions of MySQL before 3.23.26.) See [Section 13.6.1, “SQL Statements for Controlling Master Servers”](#).

- QUERY CACHE

Removes all query results from the query cache.

- SLAVE

Makes the slave forget its replication position in the master binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one. (Known as `FLUSH SLAVE` in versions of MySQL before 3.23.26.) See [Section 13.6.2, “SQL Statements for Controlling Slave Servers”](#).

13.6. Replication Statements

This section describes SQL statements related to replication. One group of statements is used for controlling master servers. The other is used for controlling slave servers.

13.6.1. SQL Statements for Controlling Master Servers

Replication can be controlled through the SQL interface. This section discusses statements for managing master replication servers. [Section 13.6.2, “SQL Statements for Controlling Slave Servers”](#), discusses statements for managing slave servers.

13.6.1.1. PURGE MASTER LOGS Syntax

```
PURGE {MASTER | BINARY} LOGS TO 'log_name'  
PURGE {MASTER | BINARY} LOGS BEFORE 'date'
```

Deletes all the binary logs listed in the log index prior to the specified log or date. The logs also are removed from the list recorded in the log index file, so that the given log becomes the first.

Example:

```
PURGE MASTER LOGS TO 'mysql-bin.010';  
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

The BEFORE variant's *date* argument can be in 'YYYY-MM-DD hh:mm:ss' format. MASTER and BINARY are synonyms.

This statement is safe to run while slaves are replicating. You do not need to stop them. If you have an active slave that currently is reading one of the logs you are trying to delete, this statement does nothing and fails with an error. However, if a slave is dormant and you happen to purge one of the logs it has yet to read, the slave will be unable to replicate after it comes up.

To safely purge logs, follow this procedure:

1. On each slave server, use `SHOW SLAVE STATUS` to check which log it is reading.
2. Obtain a listing of the binary logs on the master server with `SHOW BINARY LOGS`.
3. Determine the earliest log among all the slaves. This is the target log. If all the slaves are up to date, this is the last log on the list.
4. Make a backup of all the logs you are about to delete. (This step is optional, but always advisable.)
5. Purge all logs up to but not including the target log.

13.6.1.2. RESET MASTER Syntax

```
RESET MASTER
```

Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

13.6.1.3. SET SQL_LOG_BIN Syntax

```
SET SQL_LOG_BIN = {0|1}
```

Disables or enables binary logging for the current connection (`SQL_LOG_BIN` is a session variable) if the client has the `SUPER` privilege. The statement is refused with an error if the client does not have that privilege.

13.6.1.4. SHOW BINLOG EVENTS Syntax

```
SHOW BINLOG EVENTS  
  [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify `'log_name'`, the first binary log is displayed.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See [Section 13.2.7, “SELECT Syntax”](#).

Note: Issuing a `SHOW BINLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the binary log (which includes all statements executed by the server that modify data). As an alternative to `SHOW BINLOG EVENTS`, use the **mysqlbinlog** utility to save the binary log to a text file for later examination and analysis. See [Section 8.10, “mysqlbinlog — Utility for Processing Binary Log Files”](#).

13.6.1.5. SHOW BINARY LOGS Syntax

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in [Section 13.6.1.1, “PURGE MASTER LOGS Syntax”](#), that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name      | File_size |
+-----+-----+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+-----+-----+
```

`SHOW MASTER LOGS` is equivalent to `SHOW BINARY LOGS`. The `File_size` column is displayed as of MySQL 5.0.7.

13.6.1.6. SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

Provides status information about the binary log files of the master. Example:

```
mysql > SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.003 | 73       | test         | manual,mysql      |
+-----+-----+-----+-----+
```

13.6.1.7. SHOW SLAVE HOSTS Syntax

```
SHOW SLAVE HOSTS
```

Displays a list of replication slaves currently registered with the master. Any slave not started with the `--report-host=slave_name` option is not visible in this list.

13.6.2. SQL Statements for Controlling Slave Servers

Replication can be controlled through the SQL interface. This section discusses statements for managing slave replication servers. [Section 13.6.1, “SQL Statements for Controlling Master Servers”](#), discusses statements for managing master servers.

13.6.2.1. CHANGE MASTER TO Syntax

```
CHANGE MASTER TO master_def [, master_def] ...
```

```
master_def:
```

```
    MASTER_HOST = 'host_name'  
    | MASTER_USER = 'user_name'  
    | MASTER_PASSWORD = 'password'  
    | MASTER_PORT = port_num  
    | MASTER_CONNECT_RETRY = count  
    | MASTER_LOG_FILE = 'master_log_name'  
    | MASTER_LOG_POS = master_log_pos  
    | RELAY_LOG_FILE = 'relay_log_name'  
    | RELAY_LOG_POS = relay_log_pos  
    | MASTER_SSL = {0|1}  
    | MASTER_SSL_CA = 'ca_file_name'  
    | MASTER_SSL_CAPATH = 'ca_directory_name'  
    | MASTER_SSL_CERT = 'cert_file_name'  
    | MASTER_SSL_KEY = 'key_file_name'  
    | MASTER_SSL_CIPHER = 'cipher_list'
```

`CHANGE MASTER TO` changes the parameters that the slave server uses for connecting to and communicating with the master server. It also updates the contents of the `master.info` and `relay-log.info` files.

`MASTER_USER`, `MASTER_PASSWORD`, `MASTER_SSL`, `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_KEY`, and `MASTER_SSL_CIPHER` provide information to the slave about how to connect to its master.

The SSL options (`MASTER_SSL`, `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_KEY`, and `MASTER_SSL_CIPHER`) can be changed even on slaves that are compiled without SSL support. They are saved to the `master.info` file, but are ignored unless you use a server that has SSL support enabled.

If you don't specify a given parameter, it keeps its old value, except as indicated in the following discussion. For example, if the password to connect to your MySQL master has changed, you just need to issue these statements to tell the slave about the new password:

```
STOP SLAVE; -- if replication was running
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- if you want to restart replication
```

There is no need to specify the parameters that do not change (host, port, user, and so forth).

`MASTER_HOST` and `MASTER_PORT` are the hostname (or IP address) of the master host and its TCP/IP port. Note that if `MASTER_HOST` is equal to `localhost`, then, like in other parts of MySQL, the port number might be ignored (if Unix socket files can be used, for example).

If you specify `MASTER_HOST` or `MASTER_PORT`, the slave assumes that the master server is different from before (even if you specify a host or port value that is the same as the current value.) In this case, the old values for the master binary log name and position are considered no longer applicable, so if you do not specify `MASTER_LOG_FILE` and `MASTER_LOG_POS` in the statement, `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4` are silently appended to it.

`MASTER_LOG_FILE` and `MASTER_LOG_POS` are the coordinates at which the slave I/O thread should begin reading from the master the next time the thread starts. If you specify either of them, you cannot specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. If neither of `MASTER_LOG_FILE` or `MASTER_LOG_POS` are specified, the slave uses the last coordinates of the *slave SQL thread* before `CHANGE MASTER` was issued. This ensures that there is no discontinuity in replication, even if the slave SQL thread was late compared to the slave I/O thread, when you merely want to change, say, the password to use.

`CHANGE MASTER` *deletes all relay log files* and starts a new one, unless you

specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. In that case, relay logs are kept; the `relay_log_purge` global variable is set silently to 0.

`CHANGE MASTER` is useful for setting up a slave when you have the snapshot of the master and have recorded the log and the offset corresponding to it. After loading the snapshot into the slave, you can run `CHANGE MASTER TO MASTER_LOG_FILE='log_name_on_master', MASTER_LOG_POS=log_offset_on_master` on the slave.

The following example changes the master and master's binary log coordinates. This is used when you want to set up the slave to replicate the master:

```
CHANGE MASTER TO
  MASTER_HOST='master2.mycompany.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='big3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

The next example shows an operation that is less frequently employed. It is used when the slave has relay logs that you want it to execute again for some reason. To do this, the master need not be reachable. You need only use `CHANGE MASTER TO` and start the SQL thread (`START SLAVE SQL_THREAD`):

```
CHANGE MASTER TO
  RELAY_LOG_FILE='slave-relay-bin.006',
  RELAY_LOG_POS=4025;
```

You can even use the second operation in a non-replication setup with a standalone, non-slave server for recovery following a crash. Suppose that your server has crashed and you have restored a backup. You want to replay the server's own binary logs (not relay logs, but regular binary logs), named (for example) `myhost-bin.*`. First, make a backup copy of these binary logs in some safe place, in case you don't exactly follow the procedure below and accidentally have the server purge the binary logs. Use `SET GLOBAL relay_log_purge=0` for additional safety. Then start the server without the `--log-bin` option, Instead, use the `--replicate-same-server-id, --relay-log=myhost-bin` (to make the server believe that these regular binary logs are relay logs) and `--skip-slave-start` options. After the server starts, issue these statements:

```
CHANGE MASTER TO
  RELAY_LOG_FILE='myhost-bin.153',
  RELAY_LOG_POS=410,
  MASTER_HOST='some_dummy_string';
START SLAVE SQL_THREAD;
```

The server reads and executes its own binary logs, thus achieving crash recovery. Once the recovery is finished, run `STOP SLAVE`, shut down the server, delete the `master.info` and `relay-log.info` files, and restart the server with its original options.

Specifying the `MASTER_HOST` option (even with a dummy value) is required to make the server think it is a slave.

13.6.2.2. LOAD DATA FROM MASTER Syntax

```
LOAD DATA FROM MASTER
```

This statement takes a snapshot of the master and copies it to the slave. It updates the values of `MASTER_LOG_FILE` and `MASTER_LOG_POS` so that the slave starts replicating from the correct position. Any table and database exclusion rules specified with the `--replicate-*do-*` and `--replicate-*ignore-*` options are honored. `--replicate-rewrite-db` is *not* taken into account because a user could use this option to set up a non-unique mapping such as `--replicate-rewrite-db="db1->db3"` and `--replicate-rewrite-db="db2->db3"`, which would confuse the slave when loading tables from the master.

Use of this statement is subject to the following conditions:

- It works only for MyISAM tables. Attempting to load a non-MyISAM table results in the following error:

```
ERROR 1189 (08S01): Net error reading from master
```

- It acquires a global read lock on the master while taking the snapshot, which prevents updates on the master during the load operation.

If you are loading large tables, you might have to increase the values of `net_read_timeout` and `net_write_timeout` on both the master and slave servers. See [Section 5.2.2, “Server System Variables”](#).

Note that `LOAD DATA FROM MASTER` does *not* copy any tables from the `mysql` database. This makes it easy to have different users and privileges on the master and the slave.

To use `LOAD DATA FROM MASTER`, the replication account that is used to connect to the master must have the `RELOAD` and `SUPER` privileges on the master and the `SELECT` privilege for all master tables you want to load. All master tables for which the user does not have the `SELECT` privilege are ignored by `LOAD DATA FROM MASTER`. This is because the master hides them from the user: `LOAD DATA FROM MASTER` calls `SHOW DATABASES` to know the master databases to load, but `SHOW DATABASES` returns only databases for which the user has some privilege. See [Section 13.5.4.8, “SHOW DATABASES Syntax”](#). On the slave side, the user that issues `LOAD DATA FROM MASTER` must have privileges for dropping and creating the databases and tables that are copied.

13.6.2.3. `LOAD TABLE tbl_name FROM MASTER` Syntax

```
LOAD TABLE tbl_name FROM MASTER
```

Transfers a copy of the table from the master to the slave. This statement is implemented mainly debugging `LOAD DATA FROM MASTER` operations. To use `LOAD TABLE`, the account used for connecting to the master server must have the `RELOAD` and `SUPER` privileges on the master and the `SELECT` privilege for the master table to load. On the slave side, the user that issues `LOAD TABLE FROM MASTER` must have privileges for dropping and creating the table.

The conditions for `LOAD DATA FROM MASTER` apply here as well. For example, `LOAD TABLE FROM MASTER` works only for MyISAM tables. The timeout notes for `LOAD DATA FROM MASTER` apply as well.

13.6.2.4. `MASTER_POS_WAIT()` Syntax

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos)
```

This is actually a function, not a statement. It is used to ensure that the slave has read and executed events up to a given position in the master's binary log. See [Section 12.9.4, “Miscellaneous Functions”](#), for a full description.

13.6.2.5. `RESET SLAVE` Syntax

RESET SLAVE

RESET SLAVE makes the slave forget its replication position in the master's binary logs. This statement is meant to be used for a clean start: It deletes the master.info and relay-log.info files, all the relay logs, and starts a new relay log.

Note: All relay logs are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a STOP SLAVE statement or if the slave is highly loaded.)

Connection information stored in the master.info file is immediately reset using any values specified in the corresponding startup options. This information includes values such as master host, master port, master user, and master password. If the slave SQL thread was in the middle of replicating temporary tables when it was stopped, and RESET SLAVE is issued, these replicated temporary tables are deleted on the slave.

13.6.2.6. SET GLOBAL SQL_SLAVE_SKIP_COUNTER Syntax

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = N
```

This statement skips the next *N* events from the master. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the slave thread is not running. Otherwise, it produces an error.

13.6.2.7. SHOW SLAVE STATUS Syntax

```
SHOW SLAVE STATUS
```

This statement provides status information on essential parameters of the slave threads. If you issue this statement using the **mysql** client, you can use a \G statement terminator rather than a semicolon to obtain a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
```

```
Master_Host: localhost
Master_User: root
Master_Port: 3306
Connect_Retry: 3
Master_Log_File: gbichot-bin.005
Read_Master_Log_Pos: 79
Relay_Log_File: gbichot-relay-bin.005
Relay_Log_Pos: 548
Relay_Master_Log_File: gbichot-bin.005
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 79
Relay_Log_Space: 552
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 8
```

SHOW SLAVE STATUS returns the following fields:

- Slave_IO_State

A copy of the state field of the output of SHOW PROCESSLIST for the slave I/O thread. This tells you what the thread is doing: trying to connect to the master, waiting for events from the master, reconnecting to the master, and so on. Possible states are listed in [Section 6.3, “Replication Implementation Details”](#). It is necessary to check this field for older versions of MySQL (prior to 5.0.12) because in these versions the thread could be running while unsuccessfully trying to connect to the master; only this field makes you aware of the connection problem. The state of the SQL thread is not copied because it is simpler. If it is running, there is no problem; if it is not, you can find the error in the Last_Error field (described below).

- Master_Host

The current master host.

- Master_User

The current user used to connect to the master.

- Master_Port

The current master port.

- Connect_Retry

The current value of the --master-connect-retry option.

- Master_Log_File

The name of the master binary log file from which the I/O thread is currently reading.

- Read_Master_Log_Pos

The position up to which the I/O thread has read in the current master binary log.

- Relay_Log_File

The name of the relay log file from which the SQL thread is currently reading and executing.

- Relay_Log_Pos

The position up to which the SQL thread has read and executed in the current relay log.

- Relay_Master_Log_File

The name of the master binary log file containing the most recent event executed by the SQL thread.

- Slave_IO_Running

Whether the I/O thread is started and has connected successfully to the master. For older versions of MySQL (prior to 4.1.14 and 5.0.12) `Slave_IO_Running` is YES if the I/O thread is started, even if the slave hasn't connected to the master yet.

- `Slave_SQL_Running`

Whether the SQL thread is started.

- `Replicate_Do_DB`, `Replicate_Ignore_DB`

The lists of databases that were specified with the `--replicate-do-db` and `--replicate-ignore-db` options, if any.

- `Replicate_Do_Table`, `Replicate_Ignore_Table`,
`Replicate_Wild_Do_Table`, `Replicate_Wild_Ignore_Table`

The lists of tables that were specified with the `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table`, and `--replicate-wild-ignore-table` options, if any.

- `Last_Errno`, `Last_Error`

The error number and error message returned by the most recently executed query. An error number of 0 and message of the empty string mean “no error.” If the `Last_Error` value is not empty, it also appears as a message in the slave's error log. For example:

```
Last_Errno: 1051
Last_Error: error 'Unknown table 'z'' on query 'drop table z'
```

The message indicates that the table `z` existed on the master and was dropped there, but it did not exist on the slave, so `DROP TABLE` failed on the slave. (This might occur, for example, if you forget to copy the table to the slave when setting up replication.)

- `Skip_Counter`

The most recently used value for `SQL_SLAVE_SKIP_COUNTER`.

- `Exec_Master_Log_Pos`

The position of the last event executed by the SQL thread from the master's binary log (`Relay_Master_Log_File`). (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) in the master's binary log corresponds to (`Relay_Log_File`, `Relay_Log_Pos`) in the relay log.

- `Relay_Log_Space`

The total combined size of all existing relay logs.

- `Until_Condition`, `Until_Log_File`, `Until_Log_Pos`

The values specified in the `UNTIL` clause of the `START SLAVE` statement.

`Until_Condition` has these values:

- None if no `UNTIL` clause was specified
- Master if the slave is reading until a given position in the master's binary logs
- Relay if the slave is reading until a given position in its relay logs

`Until_Log_File` and `Until_Log_Pos` indicate the log filename and position values that define the point at which the SQL thread stops executing.

- `Master_SSL_Allowed`, `Master_SSL_CA_File`, `Master_SSL_CA_Path`, `Master_SSL_Cert`, `Master_SSL_Cipher`, `Master_SSL_Key`

These fields show the SSL parameters used by the slave to connect to the master, if any.

`Master_SSL_Allowed` has these values:

- Yes if an SSL connection to the master is permitted
- No if an SSL connection to the master is not permitted
- Ignored if an SSL connection is permitted but the slave server does not have SSL support enabled

The values of the other SSL-related fields correspond to the values of the --

master-ca, --master-capath, --master-cert, --master-cipher, and --master-key options.

- Seconds_Behind_Master

This field is an indication of how “late” the slave is:

- When the slave SQL thread is actively running (processing updates), this field is the number of seconds that have elapsed since the timestamp of the most recent event on the master executed by that thread.
- When the SQL thread has caught up to the slave I/O thread and goes idle waiting for more events from the I/O thread, this field is zero.

In essence, this field measures the time difference in seconds between the slave SQL thread and the slave I/O thread.

If the network connection between master and slave is fast, the slave I/O thread is very close to the master, so this field is a good approximation of how late the slave SQL thread is compared to the master. If the network is slow, this is *not* a good approximation; the slave SQL thread may quite often be caught up with the slow-reading slave I/O thread, so Seconds_Behind_Master often shows a value of 0, even if the I/O thread is late compared to the master. In other words, *this column is useful only for fast networks*.

This time difference computation works even though the master and slave do not have identical clocks (the clock difference is computed when the slave I/O thread starts, and assumed to remain constant from then on). Seconds_Behind_Master is NULL (which means “unknown”) if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. For example if the slave I/O thread is sleeping for the number of seconds given by the --master-connect-retry option before reconnecting, NULL is shown, as the slave cannot know what the master is doing, and so cannot say reliably how late it is.

This field has one limitation. The timestamp is preserved through replication, which means that, if a master M1 is itself a slave of M0, any

event from M1's binlog which originates in replicating an event from M0's binlog has the timestamp of that event. This enables MySQL to replicate `TIMESTAMP` successfully. However, the drawback for `Seconds_Behind_Master` is that if M1 also receives direct updates from clients, the value randomly deviates, because sometimes the last M1's event is from M0 and sometimes it is the most recent timestamp from a direct update.

13.6.2.8. `START SLAVE` Syntax

```
START SLAVE [thread_type [, thread_type] ... ]
START SLAVE [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
START SLAVE [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos
```

thread_type: `IO_THREAD` | `SQL_THREAD`

`START SLAVE` with no *thread_type* options starts both of the slave threads. The I/O thread reads queries from the master server and stores them in the relay log. The SQL thread reads the relay log and executes the queries. `START SLAVE` requires the `SUPER` privilege.

If `START SLAVE` succeeds in starting the slave threads, it returns without any error. However, even in that case, it might be that the slave threads start and then later stop (for example, because they do not manage to connect to the master or read its binary logs, or some other problem). `START SLAVE` does not warn you about this. You must check the slave's error log for error messages generated by the slave threads, or check that they are running satisfactorily with `SHOW SLAVE STATUS`.

You can add `IO_THREAD` and `SQL_THREAD` options to the statement to name which of the threads to start.

An `UNTIL` clause may be added to specify that the slave should start and run until the SQL thread reaches a given point in the master binary logs or in the slave relay logs. When the SQL thread reaches that point, it stops. If the `SQL_THREAD` option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both slave threads. If the SQL thread is running, the `UNTIL` clause is ignored and a warning is issued.

For an UNTIL clause, you must specify both a log filename and position. Do not mix master and relay log options.

Any UNTIL condition is reset by a subsequent STOP SLAVE statement, a START SLAVE statement that includes no UNTIL clause, or a server restart.

The UNTIL clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the slave replicate a statement. For example, if an unwise DROP TABLE statement was executed on the master, you can use UNTIL to tell the slave to execute up to that point but no farther. To find what the event is, use **mysqlbinlog** with the master logs or slave relay logs, or by using a SHOW BINLOG EVENTS statement.

If you are using UNTIL to have the slave process replicated queries in sections, it is recommended that you start the slave with the --skip-slave-start option to prevent the SQL thread from running when the slave server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The SHOW SLAVE STATUS statement includes output fields that display the current values of the UNTIL condition.

In old versions of MySQL (before 4.0.5), this statement was called SLAVE START. This usage is still accepted in MySQL 5.0 for backward compatibility, but is deprecated.

13.6.2.9. STOP SLAVE Syntax

```
STOP SLAVE [thread_type [, thread_type] ... ]
```

thread_type: IO_THREAD | SQL_THREAD

Stops the slave threads. STOP SLAVE requires the SUPER privilege.

Like START SLAVE, this statement may be used with the IO_THREAD and SQL_THREAD options to name the thread or threads to be stopped.

In old versions of MySQL (before 4.0.5), this statement was called SLAVE STOP. This usage is still accepted in MySQL 5.0 for backward compatibility, but is deprecated.

13.7. SQL Syntax for Prepared Statements

MySQL 5.0 provides support for server-side prepared statements. This support takes advantage of the efficient client/server binary protocol implemented in MySQL 4.1, provided that you use an appropriate client programming interface. Candidate interfaces include the MySQL C API client library (for C programs), MySQL Connector/J (for Java programs), and MySQL Connector/NET. For example, the C API provides a set of function calls that make up its prepared statement API. See [Section 22.2.4, “C API Prepared Statements”](#). Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library, one example being the [mysql_i extension](#), available in PHP 5.0 and later.

An alternative SQL interface to prepared statements is available. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.
- You can use it from any program that allows you to send SQL statements to the server to be executed, such as the **mysql** client program.
- You can use it even if the client is using an old version of the client library. The only requirement is that you be able to connect to a server that is recent enough to support SQL syntax for prepared statements.

SQL syntax for prepared statements is intended to be used for situations such as these:

- You want to test how prepared statements work in your application before coding it.
- An application has problems executing prepared statements and you want to determine interactively what the problem is.
- You want to create a test case that describes a problem you are having with prepared statements, so that you can file a bug report.

- You need to use prepared statements but do not have access to a programming API that supports them.

SQL syntax for prepared statements is based on three SQL statements:

- `PREPARE stmt_name FROM preparable_stmt`

The `PREPARE` statement prepares a statement and assigns it a name, *stmt_name*, by which to refer to the statement later. Statement names are not case sensitive. *preparable_stmt* is either a string literal or a user variable that contains the text of the statement. The text must represent a single SQL statement, not multiple statements. Within the statement, ‘?’ characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The ‘?’ characters should not be enclosed within quotes, even if you intend to bind them to string values. Parameter markers can be used only where data values should appear, not for SQL keywords, identifiers, and so forth.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

The scope of a prepared statement is the client session within which it is created. Other clients cannot see it.

- `EXECUTE stmt_name [USING @var_name [, @var_name] ...]`

After preparing a statement, you execute it with an `EXECUTE` statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a `USING` clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the `USING` clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

- {DEALLOCATE | DROP} PREPARE stmt_name

To deallocate a prepared statement, use the DEALLOCATE PREPARE statement. Attempting to execute a prepared statement after deallocating it results in an error.

If you terminate a client session without deallocating a previously prepared statement, the server deallocates it automatically.

The following SQL statements can be used in prepared statements: CREATE TABLE, DELETE, DO, INSERT, REPLACE, SELECT, SET, UPDATE, and most SHOW statements. supported. ANALYZE TABLE, OPTIMIZE TABLE, and REPAIR TABLE are supported as of MySQL 5.0.23. Other statements are not yet supported.

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypot
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|           5 |
+-----+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement as a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|          10 |
+-----+
```

```
mysql> DEALLOCATE PREPARE stmt2;
```

As of MySQL 5.0.7, placeholders can be used for the arguments of the LIMIT clause when using prepared statements. See [Section 13.2.7, “SELECT Syntax”](#).

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to PREPARE cannot itself be a PREPARE, EXECUTE, or DEALLOCATE PREPARE statement.

SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the `mysql_stmt_prepare()` C API function to prepare a PREPARE, EXECUTE, or DEALLOCATE PREPARE statement.

SQL syntax for prepared statements cannot be used within stored routines (procedures or functions), or triggers. This restriction is lifted as of MySQL 5.0.13 for stored procedures, but not for stored functions or triggers.

SQL syntax for prepared statements does not support multi-statements (that is, multiple statements within a single string separated by ‘;’ characters).

Chapter 14. Storage Engines and Table Types

Table of Contents

[14.1. The MyISAM Storage Engine](#)

[14.1.1. MyISAM Startup Options](#)

[14.1.2. Space Needed for Keys](#)

[14.1.3. MyISAM Table Storage Formats](#)

[14.1.4. MyISAM Table Problems](#)

[14.2. The InnoDB Storage Engine](#)

[14.2.1. InnoDB Overview](#)

[14.2.2. InnoDB Contact Information](#)

[14.2.3. InnoDB Configuration](#)

[14.2.4. InnoDB Startup Options and System Variables](#)

[14.2.5. Creating the InnoDB Tablespace](#)

[14.2.6. Creating and Using InnoDB Tables](#)

[14.2.7. Adding and Removing InnoDB Data and Log Files](#)

[14.2.8. Backing Up and Recovering an InnoDB Database](#)

[14.2.9. Moving an InnoDB Database to Another Machine](#)

[14.2.10. InnoDB Transaction Model and Locking](#)

[14.2.11. InnoDB Performance Tuning Tips](#)

[14.2.12. Implementation of Multi-Versioning](#)

[14.2.13. InnoDB Table and Index Structures](#)

[14.2.14. InnoDB File Space Management and Disk I/O](#)

[14.2.15. InnoDB Error Handling](#)

[14.2.16. Restrictions on InnoDB Tables](#)

[14.2.17. InnoDB Troubleshooting](#)

[14.3. The MERGE Storage Engine](#)

[14.3.1. MERGE Table Problems](#)

[14.4. The MEMORY \(HEAP\) Storage Engine](#)

[14.5. The BDB \(BerkeleyDB\) Storage Engine](#)

[14.5.1. Operating Systems Supported by BDB](#)

[14.5.2. Installing BDB](#)

[14.5.3. BDB Startup Options](#)

[14.5.4. Characteristics of BDB Tables](#)

[14.5.5. Restrictions on BDB Tables](#)

[14.5.6. Errors That May Occur When Using BDB Tables](#)

[14.6. The EXAMPLE Storage Engine](#)

[14.7. The FEDERATED Storage Engine](#)

[14.7.1. Description of the FEDERATED Storage Engine](#)

[14.7.2. How to use FEDERATED Tables](#)

[14.7.3. Limitations of the FEDERATED Storage Engine](#)

[14.8. The ARCHIVE Storage Engine](#)

[14.9. The CSV Storage Engine](#)

[14.10. The BLACKHOLE Storage Engine](#)

MySQL supports several storage engines that act as handlers for different table types. MySQL storage engines include both those that handle transaction-safe tables and those that handle non-transaction-safe tables:

- MyISAM manages non-transactional tables. It provides high-speed storage and retrieval, as well as fulltext searching capabilities. MyISAM is supported in all MySQL configurations, and is the default storage engine unless you have configured MySQL to use a different one by default.
- The MEMORY storage engine provides in-memory tables. The MERGE storage engine allows a collection of identical MyISAM tables to be handled as a single table. Like MyISAM, the MEMORY and MERGE storage engines handle non-transactional tables, and both are also included in MySQL by default.

Note: The MEMORY storage engine formerly was known as the HEAP engine.

- The InnoDB and BDB storage engines provide transaction-safe tables. BDB is included in MySQL-Max binary distributions on those operating systems that support it. InnoDB is also included by default in all MySQL 5.0 binary distributions. In source distributions, you can enable or disable either engine by configuring MySQL as you like.
- The EXAMPLE storage engine is a “stub” engine that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them. The purpose of this engine is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.
- NDB Cluster is the storage engine used by MySQL Cluster to implement tables that are partitioned over many computers. It is available in MySQL-Max 5.0 binary distributions. This storage engine is currently supported on

Linux, Solaris, and Mac OS X only. We intend to add support for this engine on other platforms, including Windows, in future MySQL releases.

- The ARCHIVE storage engine is used for storing large amounts of data without indexes with a very small footprint.
- The CSV storage engine stores data in text files using comma-separated values format.
- The BLACKHOLE storage engine accepts but does not store data and retrievals always return an empty set.
- The FEDERATED storage engine was added in MySQL 5.0.3. This engine stores data in a remote database. Currently, it works with MySQL only, using the MySQL C Client API. In future releases, we intend to enable it to connect to other data sources using other drivers or client connection methods.

This chapter describes each of the MySQL storage engines except for NDB Cluster, which is covered in [Chapter 15, MySQL Cluster](#).

When you create a new table, you can specify which storage engine to use by adding an ENGINE or TYPE table option to the CREATE TABLE statement:

```
CREATE TABLE t (i INT) ENGINE = INNODB;  
CREATE TABLE t (i INT) TYPE = MEMORY;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term and TYPE is deprecated.

If you omit the ENGINE or TYPE option, the default storage engine is used. Normally, this is MyISAM, but you can change it by using the --default-storage-engine or --default-table-type server startup option, or by setting the default-storage-engine or default-table-type option in the my.cnf configuration file.

You can set the default storage engine to be used during the current session by setting the storage_engine or table_type variable:

```
SET storage_engine=MYISAM;  
SET table_type=BDB;
```

When MySQL is installed on Windows using the MySQL Configuration Wizard, the InnoDB storage engine can be selected as the default instead of MyISAM. See [Section 2.3.4.6, “The Database Usage Dialog”](#).

To convert a table from one storage engine to another, use an ALTER TABLE statement that indicates the new engine:

```
ALTER TABLE t ENGINE = MYISAM;  
ALTER TABLE t TYPE = BDB;
```

See [Section 13.1.5, “CREATE TABLE Syntax”](#), and [Section 13.1.2, “ALTER TABLE Syntax”](#).

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine, usually MyISAM. This behavior is convenient when you want to copy tables between MySQL servers that support different storage engines. (For example, in a replication setup, perhaps your master server supports transactional storage engines for increased safety, but the slave servers use only non-transactional storage engines for greater speed.)

This automatic substitution of the default storage engine for unavailable engines can be confusing for new MySQL users. A warning is generated whenever a storage engine is automatically changed.

For new tables, MySQL always creates an .frm file to hold the table and column definitions. The table's index and data may be stored in one or more other files, depending on the storage engine. The server creates the .frm file above the storage engine level. Individual storage engines create any additional files required for the tables that they manage.

A database may contain tables of different types. That is, tables need not all be created with the same storage engine.

Transaction-safe tables (TSTs) have several advantages over non-transaction-safe tables (NTSTs):

- They are safer. Even if MySQL crashes or you get hardware problems, you can get your data back, either by automatic recovery or from a backup plus the transaction log.

- You can combine many statements and accept them all at the same time with the `COMMIT` statement (if autocommit is disabled).
- You can execute `ROLLBACK` to ignore your changes (if autocommit is disabled).
- If an update fails, all of your changes are reverted. (With non-transaction-safe tables, all changes that have taken place are permanent.)
- Transaction-safe storage engines can provide better concurrency for tables that get many updates concurrently with reads.

You can combine transaction-safe and non-transaction-safe tables in the same statements to get the best of both worlds. However, although MySQL supports several transaction-safe storage engines, for best results, you should not mix different storage engines within a transaction with autocommit disabled. For example, if you do this, changes to non-transaction-safe tables still are committed immediately and cannot be rolled back. For information about this and other problems that can occur in transactions that use mixed storage engines, see [Section 13.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

Non-transaction-safe tables have several advantages of their own, all of which occur because there is no transaction overhead:

- Much faster
- Lower disk space requirements
- Less memory required to perform updates

14.1. The MyISAM Storage Engine

MyISAM is the default storage engine. It is based on the older ISAM code but has many useful extensions. (Note that MySQL 5.0 does *not* support ISAM.)

Each MyISAM table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format. The data file has an `.MYD` (MYData) extension. The index file has an `.MYI` (MYIndex) extension.

To specify explicitly that you want a MyISAM table, indicate that with an `ENGINE` table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

The older term `TYPE` is supported as a synonym for `ENGINE` for backward compatibility, but `ENGINE` is the preferred term and `TYPE` is deprecated.

Normally, it is unnecessary to use `ENGINE` to specify the MyISAM storage engine. MyISAM is the default engine unless the default has been changed. To ensure that MyISAM is used in situations where the default might have been changed, include the `ENGINE` option explicitly.

You can check or repair MyISAM tables with the **mysqlcheck** client or **myisamchk** utility. You can also compress MyISAM tables with **myisampack** to take up much less space. See [Section 8.11, “mysqlcheck — A Table Maintenance and Repair Program”](#), [Section 5.10.4.1, “Using myisamchk for Crash Recovery”](#), and [Section 8.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#).

MyISAM tables have the following characteristics:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirements for binary portability are that the machine uses two's-complement signed integers and IEEE floating-point format. These requirements are widely used among mainstream machines. Binary compatibility might not be applicable to embedded systems, which sometimes have peculiar

processors.

There is no significant speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it takes little more processing to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- All numeric key values are stored with the high byte first to allow better index compression.
- Large files (up to 63-bit file length) are supported on filesystems and operating systems that support large files.
- There is a limit of 2^{32} (~4.295E+09) rows in a MyISAM table. You can increase this limitation if you build MySQL with the `--with-big-tables` option then the row limitation is increased to $(2^{32})^2$ (1.844E+19) rows. See [Section 2.9.2, “Typical configure Options”](#). Beginning with MySQL 5.0.4 all standard binaries are built with this option.
- The maximum number of indexes per MyISAM table is 64. This can be changed by recompiling. Beginning with MySQL 5.0.18, you can configure the build by invoking **configure** with the `--with-max-indexes=N` option, where *N* is the maximum number of indexes to permit per MyISAM table. *N* must be less than or equal to 128. Before MySQL 5.0.18, you must change the source.

The maximum number of columns per index is 16.

- The maximum key length is 1000 bytes. This can also be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.
- When rows are inserted in sorted order (as when you are using an `AUTO_INCREMENT` column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.
- Internal handling of one `AUTO_INCREMENT` column per table is supported. MyISAM automatically updates this column for `INSERT` and `UPDATE` operations. This makes `AUTO_INCREMENT` columns faster (at least 10%).

Values at the top of the sequence are not reused after being deleted. (When an `AUTO_INCREMENT` column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The `AUTO_INCREMENT` value can be reset with `ALTER TABLE` or **myisamchk**.

- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.
- If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. (These are known as concurrent inserts.) A free block can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again. See [Section 7.3.3, “Concurrent Inserts”](#).
- You can put the data file and index file on different directories to get more speed with the `DATA DIRECTORY` and `INDEX DIRECTORY` table options to `CREATE TABLE`. See [Section 13.1.5, “CREATE TABLE Syntax”](#).
- `BLOB` and `TEXT` columns can be indexed.
- `NULL` values are allowed in indexed columns. This takes 0–1 bytes per key.
- Each character column can have a different character set. See [Chapter 10, *Character Set Support*](#).
- There is a flag in the MyISAM index file that indicates whether the table was closed correctly. If **mysqld** is started with the `--myisam-recover` option, MyISAM tables are automatically checked when opened, and are repaired if the table wasn't closed properly.
- **myisamchk** marks tables as checked if you run it with the `--update-state` option. **myisamchk --fast** checks only those tables that don't have this mark.
- **myisamchk --analyze** stores statistics for portions of keys, as well as for entire keys.

- **myisampack** can pack BLOB and VARCHAR columns.

MyISAM also supports the following features:

- Support for a true VARCHAR type; a VARCHAR column starts with a length stored in one or two bytes.
- Tables with VARCHAR columns may have fixed or dynamic row length.
- The sum of the lengths of the VARCHAR and CHAR columns in a table may be up to 64KB.
- A hashed computed index can be used for UNIQUE. This allows you to have UNIQUE on any combination of columns in a table. (However, you cannot search on a UNIQUE computed index.)

Additional resources

- A forum dedicated to the MyISAM storage engine is available at <http://forums.mysql.com/list.php?21>.

14.1.1. MyISAM Startup Options

The following options to **mysqld** can be used to change the behavior of MyISAM tables. For additional information, see [Section 5.2.1, “mysqld Command Options”](#).

- `--myisam-recover=mode`

Set the mode for automatic recovery of crashed MyISAM tables.

- `--delay-key-write=ALL`

Don't flush key buffers between writes for any MyISAM table.

Note: If you do this, you should not access MyISAM tables from another program (such as from another MySQL server or with **myisamchk**) when the tables are in use. Doing so risks index corruption. Using `--external-locking` does not eliminate this risk.

The following system variables affect the behavior of MyISAM tables. For additional information, see [Section 5.2.2, “Server System Variables”](#).

- `bulk_insert_buffer_size`

The size of the tree cache used in bulk insert optimization. **Note:** This is a limit *per thread*!

- `myisam_max_extra_sort_file_size`

Used to help MySQL to decide when to use the slow but safe key cache index creation method. **Note:** This parameter was given in bytes before MySQL 5.0.6, when it was removed.

- `myisam_max_sort_file_size`

The maximum size of the temporary file that MySQL is allowed to use while re-creating a MyISAM index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

- `myisam_sort_buffer_size`

Set the size of the buffer used when recovering tables.

Automatic recovery is activated if you start **mysqld** with the `--myisam-recover` option. In this case, when the server opens a MyISAM table, it checks whether the table is marked as crashed or whether the open count variable for the table is not 0 and you are running the server with external locking disabled. If either of these conditions is true, the following happens:

- The server checks the table for errors.
- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.

- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

If the recovery wouldn't be able to recover all rows from previously completed statements and you didn't specify `FORCE` in the value of the `--myisam-recover` option, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

Note that if the automatic recovery value includes `BACKUP`, the recovery process creates files with names of the form `tbl_name-datetime.BAK`. You should have a **cron** script that automatically moves these files from the database directories to backup media.

14.1.2. Space Needed for Keys

MyISAM tables use B-tree indexes. You can roughly calculate the size for the index file as $(key_length+4)/0.67$, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if a string column has a lot of trailing space or is a `VARCHAR` column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In MyISAM tables, you can also prefix compress numbers by specifying the `PACK_KEYS=1` table option when you create the table. Numbers are stored with the high byte first, so this helps when you have many integer keys that have an identical prefix.

14.1.3. MyISAM Table Storage Formats

MyISAM supports three different storage formats. Two of them, fixed and dynamic format, are chosen automatically depending on the type of columns you are using. The third, compressed format, can be created only with the **mysampack** utility.

When you use `CREATE TABLE` or `ALTER TABLE` for a table that has no BLOB or TEXT columns, you can force the table format to `FIXED` or `DYNAMIC` with the `ROW_FORMAT` table option. This causes CHAR and VARCHAR columns to become CHAR for `FIXED` format, or VARCHAR for `DYNAMIC` format.

You can decompress tables by specifying `ROW_FORMAT=DEFAULT` with `ALTER TABLE`.

See [Section 13.1.5, “CREATE TABLE Syntax”](#), for information about `ROW_FORMAT`.

14.1.3.1. Static (Fixed-Length) Table Characteristics

Static format is the default for MyISAM tables. It is used when the table contains no variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT). Each row is stored using a fixed number of bytes.

Of the three MyISAM storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats due to the ease with which rows in the data file can be found on disk: To look up a row based on a row number in the index, multiply the row number by the row length to calculate the row position. Also, when scanning a table, it is very easy to read a constant number of rows with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format MyISAM file. In this case, **mysamchk** can easily determine where each row starts and ends, so it can usually reclaim all rows except the partially written one. Note that MyISAM table indexes can always be reconstructed based on the data rows.

Static-format tables have these characteristics:

- CHAR columns are space-padded to the column width. This is also true for NUMERIC and DECIMAL columns created before MySQL 5.0.3. BINARY columns are space-padded to the column width before MySQL 5.0.15. As

of 5.0.15, BINARY columns are padded with 0x00 bytes.

- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because rows are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of rows and want to return free disk space to the operating system. To do this, use `OPTIMIZE TABLE` or **myisamchk -r**.
- Usually require more disk space than dynamic-format tables.

14.1.3.2. Dynamic Table Characteristics

Dynamic storage format is used if a MyISAM table contains any variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT), or if the table was created with the `ROW_FORMAT=DYNAMIC` table option.

Dynamic format is a little more complex than static format because each row has a header that indicates how long it is. A row can become fragmented (stored in non-contiguous pieces) when it is made longer as a result of an update.

You can use `OPTIMIZE TABLE` or **myisamchk -r** to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

Dynamic-format tables have these characteristics:

- All string columns are dynamic except those with a length less than four.
- Each row is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). Note that this does not include columns that contain NULL values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Non-empty strings are saved as a length byte plus the string contents.

- Much less disk space usually is required than for fixed-length tables.
- Each row uses only as much space as is required. However, if a row becomes larger, it is split into as many pieces as are required, resulting in row fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run `OPTIMIZE TABLE` or **myisamchk -r** from time to time to improve performance. Use **myisamchk -ei** to obtain table statistics.
- More difficult than static-format tables to reconstruct after a crash, because rows may be fragmented into many pieces and links (fragments) may be missing.
- The expected row length for dynamic-sized rows is calculated using the following expression:

$$\begin{aligned}
 &3 \\
 &+ (\textit{number of columns} + 7) / 8 \\
 &+ (\textit{number of char columns}) \\
 &+ (\textit{packed size of numeric columns}) \\
 &+ (\textit{length of strings}) \\
 &+ (\textit{number of NULL columns} + 7) / 8
 \end{aligned}$$

There is a penalty of 6 bytes for each link. A dynamic row is linked whenever an update causes an enlargement of the row. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, another link is created. You can find the number of links using **myisamchk -ed**. All links may be removed with `OPTIMIZE TABLE` or **myisamchk -r**.

14.1.3.3. Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the **mysampack** tool. Compressed tables can be uncompressed with **myisamchk**.

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is helpful when using slow disks (such as CD-ROMs).
- Each row is compressed separately, so there is very little access overhead.

The header for a row takes up one to three bytes depending on the biggest row in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:

- Suffix space compression.
 - Prefix space compression.
 - Numbers with a value of zero are stored using one bit.
 - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from -128 to 127.
 - If a column has only a small set of possible values, the data type is converted to `ENUM`.
 - A column may use any combination of the preceding compression types.
- Can be used for fixed-length or dynamic-length rows.

Note. While a compressed table is read-only, and you cannot therefore update or add rows in the table, DDL (Data Definition Language) operations are still valid. For example, you may still use `DROP` to drop the table, and `TRUNCATE` to empty the table.

14.1.4. MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted. The following discussion describes how this can happen and how to handle it.

14.1.4.1. Corrupted MyISAM Tables

Even though the MyISAM table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get

corrupted tables if any of the following events occur:

- The **mysqld** process is killed in the middle of a write.
- An unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware failures.
- You are using an external program (such as **myisamchk**) to modify a table that is being modified by the server at the same time.
- A software bug in the MySQL or MyISAM code.

Typical symptoms of a corrupt table are:

- You get the following error while selecting data from the table:
Incorrect key file for table: '...'. Try to repair it
- Queries don't find rows in the table or return incomplete results.

You can check the health of a MyISAM table using the `CHECK TABLE` statement, and repair a corrupted MyISAM table with `REPAIR TABLE`. When **mysqld** is not running, you can also check or repair a table with the **myisamchk** command. See [Section 13.5.2.3, “CHECK TABLE Syntax”](#), [Section 13.5.2.6, “REPAIR TABLE Syntax”](#), and [Section 8.3, “myisamchk — MyISAM Table-Maintenance Utility”](#).

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of a server crash. You can verify this easily by looking for a recent restarted `mysqld` message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption may have occurred during normal operation. This is a bug. You should try to create a reproducible test case that demonstrates the problem. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#), and [Section E.1.6, “Making a Test Case If You Experience Table Corruption”](#).

14.1.4.2. Problems from Tables Not Being Closed Properly

Each MyISAM index file (.MYI file) has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from CHECK TABLE or **myisamchk**, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because a FLUSH TABLES operation was performed or because there is no room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can become incorrect only under these conditions:

- A MyISAM table is copied without first issuing LOCK TABLES and FLUSH TABLES.
- MySQL has crashed between an update and the final close. (Note that the table may still be okay, because MySQL always issues writes for everything between each statement.)
- A table was modified by **myisamchk --recover** or **myisamchk --update-state** at the same time that it was in use by **mysqld**.
- Multiple **mysqld** servers are using the table and one server performed a REPAIR TABLE or CHECK TABLE on the table while it was in use by another

server. In this setup, it is safe to use `CHECK TABLE`, although you might get the warning from other servers. However, `REPAIR TABLE` should be avoided because when one server replaces the data file with a new one, this is not known to the other servers.

In general, it is a bad idea to share a data directory among multiple servers. See [Section 5.13, “Running Multiple MySQL Servers on the Same Machine”](#), for additional discussion.

14.2. The InnoDB Storage Engine

14.2.1. InnoDB Overview

InnoDB provides MySQL with a transaction-safe (ACID compliant) storage engine that has commit, rollback, and crash recovery capabilities. InnoDB does locking on the row level and also provides an Oracle-style consistent non-locking read in SELECT statements. These features increase multi-user concurrency and performance. There is no need for lock escalation in InnoDB because row-level locks fit in very little space. InnoDB also supports FOREIGN KEY constraints. You can freely mix InnoDB tables with tables from other MySQL storage engines, even within the same statement.

InnoDB has been designed for maximum performance when processing large data volumes. Its CPU efficiency is probably not matched by any other disk-based relational database engine.

Fully integrated with MySQL Server, the InnoDB storage engine maintains its own buffer pool for caching data and indexes in main memory. InnoDB stores its tables and indexes in a tablespace, which may consist of several files (or raw disk partitions). This is different from, for example, MyISAM tables where each table is stored using separate files. InnoDB tables can be of any size even on operating systems where file size is limited to 2GB.

InnoDB is included in binary distributions by default. The Windows Essentials installer makes InnoDB the MySQL default storage engine on Windows.

InnoDB is used in production at numerous large database sites requiring high performance. The famous Internet news site Slashdot.org runs on InnoDB. Mytrix, Inc. stores over 1TB of data in InnoDB, and another site handles an average load of 800 inserts/updates per second in InnoDB.

InnoDB is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see <http://www.mysql.com/company/legal/licensing/>.

Additional resources

- A forum dedicated to the InnoDB storage engine is available at <http://forums.mysql.com/list.php?22>.

14.2.2. InnoDB Contact Information

Contact information for Innobase Oy, producer of the InnoDB engine:

Web site: <http://www.innodb.com/>
Email: <sales@innodb.com>
Phone: +358-9-6969 3250 (office)
+358-40-5617367 (mobile)

Innobase Oy Inc.
World Trade Center Helsinki
Aleksanterinkatu 17
P.O.Box 800
00101 Helsinki
Finland

14.2.3. InnoDB Configuration

The InnoDB storage engine is enabled by default. If you don't want to use InnoDB tables, you can add the `skip-innodb` option to your MySQL option file.

Note: InnoDB provides MySQL with a transaction-safe (ACID compliant) storage engine that has commit, rollback, and crash recovery capabilities. **However, it cannot do so** if the underlying operating system or hardware does not work as advertised. Many operating systems or disk subsystems may delay or reorder write operations to improve performance. On some operating systems, the very system call that should wait until all unwritten data for a file has been flushed — `fsync()` — might actually return before the data has been flushed to stable storage. Because of this, an operating system crash or a power outage may destroy recently committed data, or in the worst case, even corrupt the database because of write operations having been reordered. If data integrity is important to you, you should perform some “pull-the-plug” tests before using anything in production. On Mac OS X 10.3 and up, InnoDB uses a special `fcntl()` file flush method. Under Linux, it is advisable to **disable the write-back cache**.

On ATAPI hard disks, a command such `hdparm -W0 /dev/hda` may work to disable the write-back cache. **Beware that some drives or disk controllers may be unable to disable the write-back cache.**

Two important disk-based resources managed by the InnoDB storage engine are its tablespace data files and its log files.

Note: If you specify no InnoDB configuration options, MySQL creates an auto-extending 10MB data file named `ibdata1` and two 5MB log files named `ib_logfile0` and `ib_logfile1` in the MySQL data directory. To get good performance, you should explicitly provide InnoDB parameters as discussed in the following examples. Naturally, you should edit the settings to suit your hardware and requirements.

The examples shown here are representative. See [Section 14.2.4, “InnoDB Startup Options and System Variables”](#) for additional information about InnoDB-related configuration parameters.

To set up the InnoDB tablespace files, use the `innodb_data_file_path` option in the `[mysqld]` section of the `my.cnf` option file. On Windows, you can use `my.ini` instead. The value of `innodb_data_file_path` should be a list of one or more data file specifications. If you name more than one data file, separate them by semicolon (`;`) characters:

```
innodb_data_file_path=logfile_spec1[;logfile_spec2]...
```

For example, a setting that explicitly creates a tablespace having the same characteristics as the default is as follows:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

This setting configures a single 10MB data file named `ibdata1` that is auto-extending. No location for the file is given, so by default, InnoDB creates it in the MySQL data directory.

Sizes are specified using `M` or `G` suffix letters to indicate units of MB or GB.

A tablespace containing a fixed-size 50MB data file named `ibdata1` and a 50MB auto-extending file named `ibdata2` in the data directory can be configured like this:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The full syntax for a data file specification includes the filename, its size, and several optional attributes:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

The autoextend attribute and those following can be used only for the last data file in the innodb_data_file_path line.

If you specify the autoextend option for the last data file, InnoDB extends the data file if it runs out of free space in the tablespace. The increment is 8MB at a time by default. It can be modified by changing the innodb_autoextend_increment system variable.

If the disk becomes full, you might want to add another data file on another disk. Instructions for reconfiguring an existing tablespace are given in [Section 14.2.7, “Adding and Removing InnoDB Data and Log Files”](#).

InnoDB is not aware of the filesystem maximum file size, so be cautious on filesystems where the maximum file size is a small value such as 2GB. To specify a maximum size for an auto-extending data file, use the max attribute. The following configuration allows ibdata1 to grow up to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

InnoDB creates tablespace files in the MySQL data directory by default. To specify a location explicitly, use the innodb_data_home_dir option. For example, to use two files named ibdata1 and ibdata2 but create them in the /ibdata directory, configure InnoDB like this:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

Note: InnoDB does not create directories, so make sure that the /ibdata directory exists before you start the server. This is also true of any log file directories that you configure. Use the Unix or DOS mkdir command to create any necessary directories.

InnoDB forms the directory path for each data file by textually concatenating the value of innodb_data_home_dir to the data file name, adding a pathname

separator (slash or backslash) between values if necessary. If the `innodb_data_home_dir` option is not mentioned in `my.cnf` at all, the default value is the “dot” directory `./`, which means the MySQL data directory. (The MySQL server changes its current working directory to its data directory when it begins executing.)

If you specify `innodb_data_home_dir` as an empty string, you can specify absolute paths for the data files listed in the `innodb_data_file_path` value. The following example is equivalent to the preceding one:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoext
```

A simple `my.cnf` example. Suppose that you have a computer with 128MB RAM and one hard disk. The following example shows possible configuration parameters in `my.cnf` or `my.ini` for InnoDB, including the `autoextend` attribute. The example suits most users, both on Unix and Windows, who do not want to distribute InnoDB data files and log files onto several disks. It creates an auto-extending data file `ibdata1` and two InnoDB log files `ib_logfile0` and `ib_logfile1` in the MySQL data directory. Also, the small archived InnoDB log file `ib_arch_log_0000000000` that InnoDB creates automatically ends up in the data directory.

```
[mysqld]
# You can write your other MySQL server options here
# ...
# Data files must be able to hold your data and indexes.
# Make sure that you have enough free disk space.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory
innodb_buffer_pool_size=70M
innodb_additional_mem_pool_size=10M
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=20M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Make sure that the MySQL server has the proper access rights to create files in the data directory. More generally, the server must have access rights in any

directory where it needs to create data files or log files.

Note that data files must be less than 2GB in some filesystems. The combined size of the log files must be less than 4GB. The combined size of data files must be at least 10MB.

When you create an InnoDB tablespace for the first time, it is best that you start the MySQL server from the command prompt. InnoDB then prints the information about the database creation to the screen, so you can see what is happening. For example, on Windows, if **mysqld** is located in C:\Program Files\MySQL\MySQL Server 5.0\bin, you can start it like this:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld" --console
```

If you do not send server output to the screen, check the server's error log to see what InnoDB prints during the startup process.

See [Section 14.2.5, “Creating the InnoDB Tablespace”](#), for an example of what the information displayed by InnoDB should look like.

You can place InnoDB options in the [mysqld] group of any option file that your server reads when it starts. The locations for option files are described in [Section 4.3.2, “Using Option Files”](#).

If you installed MySQL on Windows using the installation and configuration wizards, the option file will be the my.ini file located in your MySQL installation directory. See [Section 2.3.4.14, “The Location of the my.ini File”](#).

If your PC uses a boot loader where the C: drive is not the boot drive, your only option is to use the my.ini file in your Windows directory (typically C:\WINDOWS or C:\WINNT). You can use the SET command at the command prompt in a console window to print the value of WINDIR:

```
C:\> SET WINDIR  
windir=C:\WINDOWS
```

If you want to make sure that **mysqld** reads options only from a specific file, you can use the --defaults-file option as the first option on the command line when starting the server:

```
mysqld --defaults-file=your_path_to_my_cnf
```

An advanced my.cnf example. Suppose that you have a Linux computer with 2GB RAM and three 60GB hard disks at directory paths /, /dr2 and /dr3. The following example shows possible configuration parameters in my.cnf for InnoDB.

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /ibdata/ibdata1:2000M;/dr2/ibdata/ibdata2:20
#
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
innodb_buffer_pool_size=1G
innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
innodb_log_files_in_group = 2
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
#
# Uncomment the next lines if you want to use them
#innodb_thread_concurrency=5
```

In some cases, database performance improves if all data is not placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. The example illustrates how to do this. It places the two data files on different disks and places the log files on the third disk. InnoDB fills the tablespace beginning with the first data file. You can also use raw disk partitions (raw devices) as InnoDB data files, which may speed up I/O. See [Section 14.2.3.2, “Using Raw Devices for the Shared Tablespace”](#).

Warning: On 32-bit GNU/Linux x86, you must be careful not to set memory usage too high. glibc may allow the process heap to grow over thread stacks, which crashes your server. It is a risk if the value of the following expression is close to or exceeds 2GB:

```
innodb_buffer_pool_size
```

```
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_si
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL AB binaries) and in the worst case also uses `sort_buffer_size + read_buffer_size` additional memory.

By compiling MySQL yourself, you can use up to 64GB of physical memory in 32-bit Windows. See the description for `innodb_buffer_pool_ave_mem_mb` in [Section 14.2.4, “InnoDB Startup Options and System Variables”](#).

How to tune other mysqld server parameters? The following values are typical and suit most users:

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Set key_buffer to 5 - 50% of your RAM depending on how much
# you use MyISAM tables, but keep key_buffer_size + InnoDB
# buffer pool size < 80% of your RAM
key_buffer_size=value
```

14.2.3.1. Using Per-Table Tablespaces

You can store each InnoDB table and its indexes in its own file. This feature is called “multiple tablespaces” because in effect each table has its own tablespace.

Using multiple tablespaces can be beneficial to users who want to move specific tables to separate physical disks or who wish to restore backups of single tables quickly without interrupting the use of the remaining InnoDB tables.

You can enable multiple tablespaces by adding this line to the `[mysqld]` section of `my.cnf`:

```
[mysqld]
innodb_file_per_table
```

After restarting the server, InnoDB stores each newly created table into its own file `tbl_name.ibd` in the database directory where the table belongs. This is

similar to what the MyISAM storage engine does, but MyISAM divides the table into a data file `tbl_name.MYD` and the index file `tbl_name.MYI`. For InnoDB, the data and the indexes are stored together in the `.ibd` file. The `tbl_name.frm` file is still created as usual.

If you remove the `innodb_file_per_table` line from `my.cnf` and restart the server, InnoDB creates tables inside the shared tablespace files again.

`innodb_file_per_table` affects only table creation, not access to existing tables. If you start the server with this option, new tables are created using `.ibd` files, but you can still access tables that exist in the shared tablespace. If you remove the option and restart the server, new tables are created in the shared tablespace, but you can still access any tables that were created using multiple tablespaces.

Note: InnoDB always needs the shared tablespace because it puts its internal data dictionary and undo logs there. The `.ibd` files are not sufficient for InnoDB to operate.

Note: You cannot freely move `.ibd` files between database directories as you can with MyISAM table files. This is because the table definition that is stored in the InnoDB shared tablespace includes the database name, and because InnoDB must preserve the consistency of transaction IDs and log sequence numbers.

To move an `.ibd` file and the associated table from one database to another, use a `RENAME TABLE` statement:

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

If you have a “clean” backup of an `.ibd` file, you can restore it to the MySQL installation from which it originated as follows:

1. Issue this `ALTER TABLE` statement:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

Caution: This statement deletes the current `.ibd` file.

2. Put the backup `.ibd` file back in the proper database directory.
3. Issue this `ALTER TABLE` statement:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

In this context, a “clean” .ibd file backup means:

- There are no uncommitted modifications by transactions in the .ibd file.
- There are no unmerged insert buffer entries in the .ibd file.
- Purge has removed all delete-marked index records from the .ibd file.
- **mysqld** has flushed all modified pages of the .ibd file from the buffer pool to the file.

You can make a clean backup .ibd file using the following method:

1. Stop all activity from the **mysqld** server and commit all transactions.
2. Wait until `SHOW ENGINE INNODB STATUS` shows that there are no active transactions in the database, and the main thread status of InnoDB is waiting for server activity. Then you can make a copy of the .ibd file.

Another method for making a clean copy of an .ibd file is to use the commercial **InnoDB Hot Backup** tool:

1. Use **InnoDB Hot Backup** to back up the InnoDB installation.
2. Start a second **mysqld** server on the backup and let it clean up the .ibd files in the backup.

14.2.3.2. Using Raw Devices for the Shared Tablespace

You can use raw disk partitions as data files in the shared tablespace. By using a raw disk, you can perform non-buffered I/O on Windows and on some Unix systems without filesystem overhead, which may improve performance.

When you create a new data file, you must put the keyword `newraw` immediately after the data file size in `innodb_data_file_path`. The partition must be at least as large as the size that you specify. Note that 1MB in InnoDB is 1024×1024 bytes, whereas 1MB in disk specifications usually means 1,000,000 bytes.

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

The next time you start the server, InnoDB notices the `newraw` keyword and initializes the new partition. However, do not create or change any InnoDB tables yet. Otherwise, when you next restart the server, InnoDB reinitializes the partition and your changes are lost. (As a safety measure InnoDB prevents users from modifying data when any partition with `newraw` is specified.)

After InnoDB has initialized the new partition, stop the server, change `newraw` in the data file specification to `raw`:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:5Graw;/dev/hdd2:2Graw
```

Then restart the server and InnoDB allows changes to be made.

On Windows, you can allocate a disk partition as a data file like this:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D: :10Gnewraw
```

The `//./` corresponds to the Windows syntax of `\\.\` for accessing physical drives.

When you use raw disk partitions, be sure that they have permissions that allow read and write access by the account used for running the MySQL server.

14.2.4. InnoDB Startup Options and System Variables

This section describes the InnoDB-related command options and system variables. System variables that are true or false can be enabled at server startup by naming them, or disabled by using a `skip-` prefix. For example, to enable or disable InnoDB checksums, you can use `--innodb_checksums` or `--skip-innodb_checksums` on the command line, or `innodb_checksums` or `skip-innodb_checksums` in an option file. System variables that take a numeric value can be specified as `--var_name=value` on the command line or as `var_name=value` in option files. For more information on specifying options and system variables, see [Section 4.3, “Specifying Program Options”](#). Many of the

system variables can be changed at runtime (see [Section 5.2.3.2, “Dynamic System Variables”](#)).

InnoDB command options:

- `--innodb`

Enables the InnoDB storage engine, if the server was compiled with InnoDB support. Use `--skip-innodb` to disable InnoDB.

- `--innodb_status_file`

Causes InnoDB to create a file named `<datadir>/innodb_status.<pid>` in the MySQL data directory. InnoDB periodically writes the output of `SHOW ENGINE INNODB STATUS` to this file.

InnoDB system variables:

- `innodb_additional_mem_pool_size`

The size in bytes of a memory pool InnoDB uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you need to allocate here. If InnoDB runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log. The default value is 1MB.

- `innodb_autoextend_increment`

The increment size (in MB) for extending the size of an auto-extending tablespace when it becomes full. The default value is 8.

- `innodb_buffer_pool_ave_mem_mb`

The size of the buffer pool (in MB), if it is placed in the AWE memory. This is relevant only in 32-bit Windows. If your 32-bit Windows operating system supports more than 4GB memory, using so-called “Address Windowing Extensions,” you can allocate the InnoDB buffer pool into the AWE physical memory using this variable. The maximum possible value for this variable is 63000. If it is greater than 0, `innodb_buffer_pool_size`

is the window in the 32-bit address space of **mysqld** where InnoDB maps that AWE memory. A good value for `innodb_buffer_pool_size` is 500MB.

To take advantage of AWE memory, you will need to recompile MySQL yourself. The current project settings needed for doing this can be found in the `innobase/os/os@proj.c` source file.

- `innodb_buffer_pool_size`

The size in bytes of the memory buffer InnoDB uses to cache data and indexes of its tables. The larger you set this value, the less disk I/O is needed to access data in tables. On a dedicated database server, you may set this to up to 80% of the machine physical memory size. However, do not set it too large because competition for physical memory might cause paging in the operating system.

- `innodb_checksums`

InnoDB can use checksum validation on all pages read from the disk to ensure extra fault tolerance against broken hardware or data files. This validation is enabled by default. However, under some rare circumstances (such as when running benchmarks) this extra safety feature is unneeded and can be disabled with `--skip-innodb_checksums`. This variable was added in MySQL 5.0.3.

- `innodb_commit_concurrency`

The number of threads that can commit at the same time. A value of 0 disables concurrency control. This variable was added in MySQL 5.0.12.

- `innodb_concurrency_tickets`

The number of threads that can enter InnoDB concurrently is determined by the `innodb_thread_concurrency` variable. A thread is placed in a queue when it tries to enter InnoDB if the number of threads has already reached the concurrency limit. When a thread is allowed to enter InnoDB, it is given a number of “free tickets” equal to the value of `innodb_concurrency_tickets`, and the thread can enter and leave InnoDB freely until it has used up its tickets. After that point, the thread again becomes subject to the concurrency check (and possible queuing) the next

time it tries to enter InnoDB. This variable was added in MySQL 5.0.3.

- `innodb_data_file_path`

The paths to individual data files and their sizes. The full directory path to each data file is formed by concatenating `innodb_data_home_dir` to each path specified here. The file sizes are specified in MB or GB (1024MB) by appending `M` or `G` to the size value. The sum of the sizes of the files must be at least 10MB. If you do not specify `innodb_data_file_path`, the default behavior is to create a single 10MB auto-extending data file named `ibdata1`. The size limit of individual files is determined by your operating system. You can set the file size to more than 4GB on those operating systems that support big files. You can also use raw disk partitions as data files. See [Section 14.2.3.2, “Using Raw Devices for the Shared Tablespace”](#).

- `innodb_data_home_dir`

The common part of the directory path for all InnoDB data files. If you do not set this value, the default is the MySQL data directory. You can specify the value as an empty string, in which case you can use absolute file paths in `innodb_data_file_path`.

- `innodb_doublewrite`

By default, InnoDB stores all data twice, first to the doublewrite buffer, and then to the actual data files. This variable is enabled by default. It can be turned off with `--skip-innodb_doublewrite` for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures. This variable was added in MySQL 5.0.3.

- `innodb_fast_shutdown`

If you set this variable to 0, InnoDB does a full purge and an insert buffer merge before a shutdown. These operations can take minutes, or even hours in extreme cases. If you set this variable to 1, InnoDB skips these operations at shutdown. The default value is 1. If you set it to 2, InnoDB will just flush its logs and then shut down cold, as if MySQL had crashed; no committed transaction will be lost, but crash recovery will be done at the next startup. The value of 2 can be used as of MySQL 5.0.5, except that it cannot be

used on NetWare.

- `innodb_file_io_threads`

The number of file I/O threads in InnoDB. Normally, this should be left at the default value of 4, but disk I/O on Windows may benefit from a larger number. On Unix, increasing the number has no effect; InnoDB always uses the default value.

- `innodb_file_per_table`

If this variable is enabled, InnoDB creates each new table using its own `.ibd` file for storing data and indexes, rather than in the shared tablespace. The default is to create tables in the shared tablespace. See [Section 14.2.3.1, “Using Per-Table Tablespaces”](#).

- `innodb_flush_log_at_trx_commit`

When `innodb_flush_log_at_trx_commit` is set to 0, the log buffer is written out to the log file once per second and the flush to disk operation is performed on the log file, but nothing is done at a transaction commit. When this value is 1 (the default), the log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file. When set to 2, the log buffer is written out to the file at each commit, but the flush to disk operation is not performed on it. However, the flushing on the log file takes place once per second also when the value is 2. Note that the once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues.

The default value of this variable is 1, which is the value that is required for ACID compliance. You can achieve better performance by setting the value different from 1, but then you can lose at most one second worth of transactions in a crash. If you set the value to 0, then any **mysqld** process crash can erase the last second of transactions. If you set the value to 2, then only an operating system crash or a power outage can erase the last second of transactions. However, InnoDB's crash recovery is not affected and thus crash recovery does work regardless of the value. Note that many operating systems and some disk hardware fool the flush-to-disk operation. They may tell **mysqld** that the flush has taken place, even though it has not. Then the durability of transactions is not guaranteed even with the setting 1, and in

the worst case a power outage can even corrupt the InnoDB database. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try using the Unix command **hdparm** to disable the caching of disk writes in hardware caches, or use some other command specific to the hardware vendor.

- `innodb_flush_method`

If set to `fdatasync` (the default), InnoDB uses `fsync()` to flush both the data and log files. If set to `O_DSYNC`, InnoDB uses `O_SYNC` to open and flush the log files, but uses `fsync()` to flush the data files. If `O_DIRECT` is specified (available on some GNU/Linux versions), InnoDB uses `O_DIRECT` to open the data files, and uses `fsync()` to flush both the data and log files. Note that InnoDB uses `fsync()` instead of `fdatasync()`, and it does not use `O_DSYNC` by default because there have been problems with it on many varieties of Unix. This variable is relevant only for Unix. On Windows, the flush method is always `async_unbuffered` and cannot be changed.

- `innodb_force_recovery`

The crash recovery mode. Warning: This variable should be set greater than 0 only in an emergency situation when you want to dump your tables from a corrupt database! Possible values are from 1 to 6. The meanings of these values are described in [Section 14.2.8.1, “Forcing InnoDB Recovery”](#). As a safety measure, InnoDB prevents any changes to its data when this variable is greater than 0.

- `innodb_lock_wait_timeout`

The timeout in seconds an InnoDB transaction may wait for a lock before being rolled back. InnoDB automatically detects transaction deadlocks in its own lock table and rolls back the transaction. InnoDB notices locks set using the `LOCK TABLES` statement. The default is 50 seconds.

Note: For the greatest possible durability and consistency in a replication setup using InnoDB with transactions, you should use `innodb_flush_log_at_trx_commit=1`, `sync_binlog=1`, and, before MySQL 5.0.3, `innodb_safe_binlog` in your master server `my.cnf` file. (`innodb_safe_binlog` is not needed from 5.0.3 on.)

- `innodb_locks_unsafe_for_binlog`

This variable controls next-key locking in InnoDB searches and index scans. By default, this variable is 0 (disabled), which means that next-key locking is enabled.

Normally, InnoDB uses an algorithm called *next-key locking*. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on any index records it encounters. Thus, the row-level locks are actually index record locks. The locks that InnoDB sets on index records also affect the “gap” preceding that index record. If a user has a shared or exclusive lock on record *R* in an index, another user cannot insert a new index record immediately before *R* in the order of the index. Enabling this variable causes InnoDB not to use next-key locking in searches or index scans. Next-key locking is still used to ensure foreign key constraints and duplicate key checking. Note that enabling this variable may cause phantom problems: Suppose that you want to read and lock all children from the `child` table with an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

Suppose that there is an index on the `id` column. The query scans that index starting from the first record where `id` is greater than 100. If the locks set on the index records do not lock out inserts made in the gaps, another client can insert a new row into the table. If you execute the same `SELECT` within the same transaction, you see a new row in the result set returned by the query. This also means that if new items are added to the database, InnoDB does not guarantee serializability. Therefore, if this variable is enabled InnoDB guarantees at most isolation level `READ COMMITTED`. (Conflict serializability is still guaranteed.)

Starting from MySQL 5.0.2, this option is even more unsafe. InnoDB in an `UPDATE` or a `DELETE` only locks rows that it updates or deletes. This greatly reduces the probability of deadlocks, but they can happen. Note that enabling this variable still does not allow operations such as `UPDATE` to overtake other similar operations (such as another `UPDATE`) even in the case when they affect different rows. Consider the following example, beginning with this table:

```
CREATE TABLE A(A INT NOT NULL, B INT) ENGINE = InnoDB;  
INSERT INTO A VALUES (1,2),(2,3),(3,2),(4,3),(5,2);  
COMMIT;
```

Suppose that one client executes these statements:

```
SET AUTOCOMMIT = 0;  
UPDATE A SET B = 5 WHERE B = 3;
```

Then suppose that another client executes these statements following those of the first client:

```
SET AUTOCOMMIT = 0;  
UPDATE A SET B = 4 WHERE B = 2;
```

In this case, the second UPDATE must wait for a commit or rollback of the first UPDATE. The first UPDATE has an exclusive lock on row (2,3), and the second UPDATE while scanning rows also tries to acquire an exclusive lock for the same row, which it cannot have. This is because UPDATE two first acquires an exclusive lock on a row and then determines whether the row belongs to the result set. If not, it releases the unnecessary lock, when the `innodb_locks_unsafe_for_binlog` variable is enabled.

Therefore, InnoDB executes UPDATE one as follows:

```
x-lock(1,2)  
unlock(1,2)  
x-lock(2,3)  
update(2,3) to (2,5)  
x-lock(3,2)  
unlock(3,2)  
x-lock(4,3)  
update(4,3) to (4,5)  
x-lock(5,2)  
unlock(5,2)
```

InnoDB executes UPDATE two as follows:

```
x-lock(1,2)  
update(1,2) to (1,4)  
x-lock(2,3) - wait for query one to commit or rollback
```

- `innodb_log_arch_dir`

The directory where fully written log files would be archived if we used log archiving. If used, the value of this variable should be set the same as `innodb_log_group_home_dir`. However, it is not required.

- `innodb_log_archive`

Whether to log InnoDB archive files. This variable is present for historical reasons, but is unused. Recovery from a backup is done by MySQL using its own log files, so there is no need to archive InnoDB log files. The default for this variable is 0.

- `innodb_log_buffer_size`

The size in bytes of the buffer that InnoDB uses to write to the log files on disk. Sensible values range from 1MB to 8MB. The default is 1MB. A large log buffer allows large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have big transactions, making the log buffer larger saves disk I/O.

- `innodb_log_file_size`

The size in bytes of each log file in a log group. The combined size of log files must be less than 4GB on 32-bit computers. The default is 5MB. Sensible values range from 1MB to $1/N$ -th of the size of the buffer pool, where N is the number of log files in the group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But larger log files also mean that recovery is slower in case of a crash.

- `innodb_log_files_in_group`

The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default (and recommended) is 2.

- `innodb_log_group_home_dir`

The directory path to the InnoDB log files. It must have the same value as `innodb_log_arch_dir`. If you do not specify any InnoDB log variables, the default is to create two 5MB files names `ib_logfile0` and `ib_logfile1` in the MySQL data directory.

- `innodb_max_dirty_pages_pct`

This is an integer in the range from 0 to 100. The default is 90. The main thread in InnoDB tries to write pages from the buffer pool so that the percentage of dirty (not yet written) pages will not exceed this value.

- `innodb_max_purge_lag`

This variable controls how to delay INSERT, UPDATE and DELETE operations when the purge operations are lagging (see [Section 14.2.12, “Implementation of Multi-Versioning”](#)). The default value of this variable is 0, meaning that there are no delays.

The InnoDB transaction system maintains a list of transactions that have delete-marked index records by UPDATE or DELETE operations. Let the length of this list be *purge_lag*. When *purge_lag* exceeds `innodb_max_purge_lag`, each INSERT, UPDATE and DELETE operation is delayed by $((\textit{purge_lag}/\textit{innodb_max_purge_lag}) \times 10) - 5$ milliseconds. The delay is computed in the beginning of a purge batch, every ten seconds. The operations are not delayed if purge cannot run because of an old consistent read view that could see the rows to be purged.

A typical setting for a problematic workload might be 1 million, assuming that our transactions are small, only 100 bytes in size, and we can allow 100MB of unpurged rows in our tables.

- `innodb_mirrored_log_groups`

The number of identical copies of log groups to keep for the database. Currently, this should be set to 1.

- `innodb_open_files`

This variable is relevant only if you use multiple tablespaces in InnoDB. It specifies the maximum number of `.ibd` files that InnoDB can keep open at one time. The minimum value is 10. The default is 300.

The file descriptors used for `.ibd` files are for InnoDB only. They are independent of those specified by the `--open-files-limit` server option, and do not affect the operation of the table cache.

- `innodb_safe_binlog`

Adds consistency guarantees between the content of InnoDB tables and the binary log. See [Section 5.12.3, “The Binary Log”](#). This variable was removed in MySQL 5.0.3, having been made obsolete by the introduction of XA transaction support.

- `innodb_support_xa`

When set to `ON` or `1` (the default), this variable enables InnoDB support for two-phase commit in XA transactions. Enabling `innodb_support_xa` causes an extra disk flush for transaction preparation. If you don't care about using XA, you can disable this variable by setting it to `OFF` or `0` to reduce the number of disk flushes and get better InnoDB performance. This variable was added in MySQL 5.0.3.

- `innodb_sync_spin_loops`

The number of times a thread waits for an InnoDB mutex to be freed before the thread is suspended. This variable was added in MySQL 5.0.3.

- `innodb_table_locks`

InnoDB honors `LOCK TABLES`; MySQL does not return from `LOCK TABLE . . . WRITE` until all other threads have released all their locks to the table. The default value is `1`, which means that `LOCK TABLES` causes InnoDB to lock a table internally. In applications using `AUTOCOMMIT=1`, InnoDB's internal table locks can cause deadlocks. You can set `innodb_table_locks=0` in the server option file to remove that problem.

- `innodb_thread_concurrency`

InnoDB tries to keep the number of operating system threads concurrently inside InnoDB less than or equal to the limit given by this variable. If you have performance issues, and `SHOW ENGINE INNODB STATUS` reveals many threads waiting for semaphores, you may have thread “thrashing” and should try setting this variable lower or higher. If you have a computer with many processors and disks, you can try setting the value higher to make better use of your computer's resources. A recommended value is the sum of the number of processors and disks your system has. greater disables

concurrency checking.

The range of this variable is 0 to 1000. A value of 20 or higher is interpreted as infinite concurrency before MySQL 5.0.19. From 5.0.19 on, 0 is interpreted as infinite. Infinite means that concurrency checking is disabled and the possibly considerable overhead of acquiring and releasing a mutex is avoided.

The default value has changed several times: 8 before MySQL 5.0.8, 20 (infinite) from 5.0.8 through 5.0.18, 0 (infinite) from 5.0.19 to 5.0.20, and 8 (finite) from 5.0.21 on.

- `innodb_thread_sleep_delay`

How long InnoDB threads sleep before joining the InnoDB queue, in microseconds. The default value is 10,000. A value of 0 disables sleep. This variable was added in MySQL 5.0.3.

- `sync_binlog`

If the value of this variable is positive, the MySQL server synchronizes its binary log to disk (`fsync()`) after every `sync_binlog` writes to this binary log. Note that there is one write to the binary log per statement if in autocommit mode, and otherwise one write per transaction. The default value is 0 which does no synchronizing to disk. A value of 1 is the safest choice, because in the event of a crash you lose at most one statement/transaction from the binary log; however, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

14.2.5. Creating the InnoDB Tablespace

Suppose that you have installed MySQL and have edited your option file so that it contains the necessary InnoDB configuration parameters. Before starting MySQL, you should verify that the directories you have specified for InnoDB data files and log files exist and that the MySQL server has access rights to those directories. InnoDB does not create directories, only files. Check also that you have enough disk space for the data and log files.

It is best to run the MySQL server **mysqld** from the command prompt when you first start the server with InnoDB enabled, not from the **mysqld_safe** wrapper or as a Windows service. When you run from a command prompt you see what **mysqld** prints and what is happening. On Unix, just invoke **mysqld**. On Windows, use the `--console` option.

When you start the MySQL server after initially configuring InnoDB in your option file, InnoDB creates your data files and log files, and prints something like this:

```
InnoDB: The first specified datafile /home/heikki/data/ibdata1
did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file /home/heikki/data/ibdata1 size to 134217728
InnoDB: Database physically writes the file full: wait...
InnoDB: datafile /home/heikki/data/ibdata2 did not exist:
new to be created
InnoDB: Setting file /home/heikki/data/ibdata2 size to 262144000
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file /home/heikki/data/logs/ib_logfile0 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile0 size
to 5242880
InnoDB: Log file /home/heikki/data/logs/ib_logfile1 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile1 size
to 5242880
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
InnoDB: Started
mysqld: ready for connections
```

At this point InnoDB has initialized its tablespace and log files. You can connect to the MySQL server with the usual MySQL client programs like **mysql**. When you shut down the MySQL server with **mysqladmin shutdown**, the output is like this:

```
010321 18:33:34 mysqld: Normal shutdown
010321 18:33:34 mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

You can look at the data file and log directories and you see the files created

there. The log directory also contains a small file named `ib_arch_log_0000000000`. That file resulted from the database creation, after which InnoDB switched off log archiving. When MySQL is started again, the data files and log files have been created already, so the output is much briefer:

```
InnoDB: Started
mysqld: ready for connections
```

If you add the `innodb_file_per_table` option to `my.cnf`, InnoDB stores each table in its own `.ibd` file in the same MySQL database directory where the `.frm` file is created. See [Section 14.2.3.1, “Using Per-Table Tablespaces”](#).

14.2.5.1. Dealing with InnoDB Initialization Problems

If InnoDB prints an operating system error during a file operation, usually the problem has one of the following causes:

- You did not create the InnoDB data file directory or the InnoDB log directory.
- **mysqld** does not have access rights to create files in those directories.
- **mysqld** cannot read the proper `my.cnf` or `my.ini` option file, and consequently does not see the options that you specified.
- The disk is full or a disk quota is exceeded.
- You have created a subdirectory whose name is equal to a data file that you specified, so the name cannot be used as a filename.
- There is a syntax error in the `innodb_data_home_dir` or `innodb_data_file_path` value.

If something goes wrong when InnoDB attempts to initialize its tablespace or its log files, you should delete all files created by InnoDB. This means all `ibdata` files and all `ib_logfile` files. In case you have already created some InnoDB tables, delete the corresponding `.frm` files for these tables (and any `.ibd` files if you are using multiple tablespaces) from the MySQL database directories as well. Then you can try the InnoDB database creation again. It is best to start the MySQL server from a command prompt so that you see what is happening.

14.2.6. Creating and Using InnoDB Tables

To create an InnoDB table, specify an `ENGINE = InnoDB` option in the `CREATE TABLE` statement:

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB
```

The older term `TYPE` is supported as a synonym for `ENGINE` for backward compatibility, but `ENGINE` is the preferred term and `TYPE` is deprecated.

The statement creates a table and an index on column `a` in the InnoDB tablespace that consists of the data files that you specified in `my.cnf`. In addition, MySQL creates a file `customers.frm` in the `test` directory under the MySQL database directory. Internally, InnoDB adds an entry for the table to its own data dictionary. The entry includes the database name. For example, if `test` is the database in which the `customers` table is created, the entry is for `'test/customers'`. This means you can create a table of the same name `customers` in some other database, and the table names do not collide inside InnoDB.

You can query the amount of free space in the InnoDB tablespace by issuing a `SHOW TABLE STATUS` statement for any InnoDB table. The amount of free space in the tablespace appears in the `Comment` section in the output of `SHOW TABLE STATUS`. For example:

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

Note that the statistics `SHOW` displays for InnoDB tables are only approximate. They are used in SQL optimization. Table and index reserved sizes in bytes are accurate, though.

14.2.6.1. How to Use Transactions in InnoDB with Different APIs

By default, each client that connects to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement as you execute it. To use multiple-statement transactions, you can switch autocommit off with the SQL statement `SET AUTOCOMMIT = 0` and use `COMMIT` and `ROLLBACK` to commit or roll back your transaction. If you want to leave autocommit on, you can enclose your transactions within `START TRANSACTION` and either `COMMIT` or `ROLLBACK`. The following example shows two transactions.

The first is committed; the second is rolled back.

```
shell> mysql test
```

```
mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
-> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM CUSTOMER;
+-----+-----+
| A     | B           |
+-----+-----+
|  10  | Heikki     |
+-----+-----+
1 row in set (0.00 sec)
mysql>
```

In APIs such as PHP, Perl DBI, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as COMMIT to the MySQL server as strings just like any other SQL statements such as SELECT or INSERT. Some APIs also offer separate special transaction commit and rollback functions or methods.

14.2.6.2. Converting MyISAM Tables to InnoDB

Important: Do not convert MySQL system tables in the mysql database (such as user or host) to the InnoDB type. This is an unsupported operation. The system tables must always be of the MyISAM type.

If you want all your (non-system) tables to be created as InnoDB tables, you can simply add the line `default-storage-engine=innodb` to the `[mysqld]` section of your server option file.

InnoDB does not have a special optimization for separate index creation the way the MyISAM storage engine does. Therefore, it does not pay to export and import the table and create indexes afterward. The fastest way to alter a table to InnoDB is to do the inserts directly to an InnoDB table. That is, use `ALTER TABLE ... ENGINE=INNODB`, or create an empty InnoDB table with identical definitions and insert the rows with `INSERT INTO ... SELECT * FROM`

If you have `UNIQUE` constraints on secondary keys, you can speed up a table import by turning off the uniqueness checks temporarily during the import operation:

```
SET UNIQUE_CHECKS=0;
... import operation ...
SET UNIQUE_CHECKS=1;
```

For big tables, this saves a lot of disk I/O because InnoDB can then use its insert buffer to write secondary index records as a batch. Be certain that the data contains no duplicate keys. `UNIQUE_CHECKS` allows but does not require storage engines to ignore duplicate keys.

To get better control over the insertion process, it might be good to insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
  WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records have been inserted, you can rename the tables.

During the conversion of big tables, you should increase the size of the InnoDB buffer pool to reduce disk I/O. Do not use more than 80% of the physical memory, though. You can also increase the sizes of the InnoDB log files.

Make sure that you do not fill up the tablespace: InnoDB tables require a lot more disk space than MyISAM tables. If an `ALTER TABLE` operation runs out of space, it starts a rollback, and that can take hours if it is disk-bound. For inserts, InnoDB uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. For rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for

millions of disk I/O operations to complete. For the complete procedure, see [Section 14.2.8.1, “Forcing InnoDB Recovery”](#).

14.2.6.3. How AUTO_INCREMENT Columns Work in InnoDB

If you specify an AUTO_INCREMENT column for an InnoDB table, the table handle in the InnoDB data dictionary contains a special counter called the auto-increment counter that is used in assigning new values for the column. This counter is stored only in main memory, not on disk.

InnoDB uses the following algorithm to initialize the auto-increment counter for a table *T* that contains an AUTO_INCREMENT column named *ai_col*: After a server startup, for the first insert into a table *T*, InnoDB executes the equivalent of this statement:

```
SELECT MAX(ai_col) FROM T FOR UPDATE;
```

InnoDB increments by one the value retrieved by the statement and assigns it to the column and to the auto-increment counter for the table. If the table is empty, InnoDB uses the value 1. If a user invokes a SHOW TABLE STATUS statement that displays output for the table *T* and the auto-increment counter has not been initialized, InnoDB initializes but does not increment the value and stores it for use by later inserts. Note that this initialization uses a normal exclusive-locking read on the table and the lock lasts to the end of the transaction.

InnoDB follows the same procedure for initializing the auto-increment counter for a freshly created table.

After the auto-increment counter has been initialized, if a user does not explicitly specify a value for an AUTO_INCREMENT column, InnoDB increments the counter by one and assigns the new value to the column. If the user inserts a row that explicitly specifies the column value, and the value is bigger than the current counter value, the counter is set to the specified column value.

You may see gaps in the sequence of values assigned to the AUTO_INCREMENT column if you roll back transactions that have generated numbers using the counter.

If a user specifies NULL or 0 for the AUTO_INCREMENT column in an INSERT,

InnoDB treats the row as if the value had not been specified and generates a new value for it.

The behavior of the auto-increment mechanism is not defined if a user assigns a negative value to the column or if the value becomes bigger than the maximum integer that can be stored in the specified integer type.

When accessing the auto-increment counter, InnoDB uses a special table-level AUTO-INC lock that it keeps to the end of the current SQL statement, not to the end of the transaction. The special lock release strategy was introduced to improve concurrency for inserts into a table containing an AUTO_INCREMENT column. Nevertheless, two transactions cannot have the AUTO-INC lock on the same table simultaneously, which can have a performance impact if the AUTO-INC lock is held for a long time. That might be the case for a statement such as `INSERT INTO t1 ... SELECT ... FROM t2` that inserts all rows from one table into another.

InnoDB uses the in-memory auto-increment counter as long as the server runs. When the server is stopped and restarted, InnoDB reinitializes the counter for each table for the first INSERT to the table, as described earlier.

Beginning with MySQL 5.0.3, InnoDB supports the `AUTO_INCREMENT = N` table option in `CREATE TABLE` and `ALTER TABLE` statements, to set the initial counter value or alter the current counter value. The effect of this option is canceled by a server restart, for reasons discussed earlier in this section.

14.2.6.4. FOREIGN KEY Constraints

InnoDB also supports foreign key constraints. The syntax for a foreign key constraint definition in InnoDB looks like this:

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
  REFERENCES tbl_name (index_col_name, ...)
  [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
  [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Foreign keys definitions are subject to the following conditions:

- Both tables must be InnoDB tables and they must not be TEMPORARY tables.

- In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist.
- In the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.
- Index prefixes on foreign key columns are not supported. One consequence of this is that BLOB and TEXT columns cannot be included in a foreign key, because indexes on those columns must always include a prefix length.
- If the CONSTRAINT *symbol* clause is given, the *symbol* value must be unique in the database. If the clause is not given, InnoDB creates the name automatically.

InnoDB rejects any INSERT or UPDATE operation that attempts to create a foreign key value in a child table if there is no a matching candidate key value in the parent table. The action InnoDB takes for any UPDATE or DELETE operation that attempts to update or delete a candidate key value in the parent table that has some matching rows in the child table is dependent on the *referential action* specified using ON UPDATE and ON DELETE subclauses of the FOREIGN KEY clause. When the user attempts to delete or update a row from a parent table, and there are one or more matching rows in the child table, InnoDB supports five options regarding the action to be taken:

- CASCADE: Delete or update the row from the parent table and automatically delete or update the matching rows in the child table. Both ON DELETE CASCADE and ON UPDATE CASCADE are supported. Between two tables, you should not define several ON UPDATE CASCADE clauses that act on the same column in the parent table or in the child table.
- SET NULL: Delete or update the row from the parent table and set the foreign key column or columns in the child table to NULL. This is valid only if the foreign key columns do not have the NOT NULL qualifier specified. Both ON DELETE SET NULL and ON UPDATE SET NULL clauses are supported.
- NO ACTION: In standard SQL, NO ACTION means *no action* in the sense that an attempt to delete or update a primary key value is not allowed to proceed if there is a related foreign key value in the referenced table. InnoDB rejects the delete or update operation for the parent table.

- **RESTRICT:** Rejects the delete or update operation for the parent table. `NO ACTION` and `RESTRICT` are the same as omitting the `ON DELETE` or `ON UPDATE` clause. (Some database systems have deferred checks, and `NO ACTION` is a deferred check. In MySQL, foreign key constraints are checked immediately, so `NO ACTION` and `RESTRICT` are the same.)
- **SET DEFAULT:** This action is recognized by the parser, but InnoDB rejects table definitions containing `ON DELETE SET DEFAULT` or `ON UPDATE SET DEFAULT` clauses.

Note that InnoDB supports foreign key references within a table. In these cases, “child table records” really refers to dependent records within the same table.

InnoDB requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. The index on the foreign key is created automatically. This is in contrast to some older versions, in which indexes had to be created explicitly or the creation of foreign key constraints would fail.

Corresponding columns in the foreign key and the referenced key must have similar internal data types inside InnoDB so that they can be compared without a type conversion. *The size and sign of integer types must be the same.* The length of string types need not be the same. If you specify a `SET NULL` action, *make sure that you have not declared the columns in the child table as NOT NULL.*

If MySQL reports an error number 1005 from a `CREATE TABLE` statement, and the error message refers to `errno 150`, table creation failed because a foreign key constraint was not correctly formed. Similarly, if an `ALTER TABLE` fails and it refers to `errno 150`, that means a foreign key definition would be incorrectly formed for the altered table. You can use `SHOW ENGINE INNODB STATUS` to display a detailed explanation of the most recent InnoDB foreign key error in the server.

Note: InnoDB does not check foreign key constraints on those foreign key or referenced key values that contain a `NULL` column.

Note: Currently, triggers are not activated by cascaded foreign key actions.

Deviation from SQL standards: If there are several rows in the parent table that have the same referenced key value, InnoDB acts in foreign key checks as if the

other parent rows with the same key value do not exist. For example, if you have defined a RESTRICT type constraint, and there is a child row with several parent rows, InnoDB does not allow the deletion of any of those parent rows.

InnoDB performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

Deviation from SQL standards: A FOREIGN KEY constraint that references a non-UNIQUE key is not standard SQL. It is an InnoDB extension to standard SQL.

Deviation from SQL standards: If ON UPDATE CASCADE or ON UPDATE SET NULL recurses to update the *same table* it has previously updated during the cascade, it acts like RESTRICT. This means that you cannot use self-referential ON UPDATE CASCADE or ON UPDATE SET NULL operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential ON DELETE SET NULL, on the other hand, is possible, as is a self-referential ON DELETE CASCADE. Cascading operations may not be nested more than 15 levels deep.

Deviation from SQL standards: Like MySQL in general, in an SQL statement that inserts, deletes, or updates many rows, InnoDB checks UNIQUE and FOREIGN KEY constraints row-by-row. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. Until InnoDB implements deferred constraint checking, some things will be impossible, such as deleting a record that refers to itself via a foreign key.

Here is a simple example that relates parent and child tables through a single-column foreign key:

```
CREATE TABLE parent (id INT NOT NULL,  
                    PRIMARY KEY (id)  
                    ) ENGINE=INNODB;  
CREATE TABLE child (id INT, parent_id INT,  
                   INDEX par_ind (parent_id),  
                   FOREIGN KEY (parent_id) REFERENCES parent(id)  
                   ON DELETE CASCADE  
                   ) ENGINE=INNODB;
```

A more complex example in which a product_order table has foreign keys for two other tables. One foreign key references a two-column index in the product table. The other references a single-column index in the customer table:

```

CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
                        price DECIMAL,
                        PRIMARY KEY(category, id)) ENGINE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
                        PRIMARY KEY (id)) ENGINE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                              product_category INT NOT NULL,
                              product_id INT NOT NULL,
                              customer_id INT NOT NULL,
                              PRIMARY KEY(no),
                              INDEX (product_category, product_id),
                              FOREIGN KEY (product_category, product_id)
                                REFERENCES product(category, id)
                                ON UPDATE CASCADE ON DELETE RESTRICT,
                              INDEX (customer_id),
                              FOREIGN KEY (customer_id)
                                REFERENCES customer(id)) ENGINE=INNODB

```

InnoDB allows you to add a new foreign key constraint to a table by using ALTER TABLE:

```

ALTER TABLE tbl_name
  ADD [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
  REFERENCES tbl_name (index_col_name, ...)
  [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
  [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]

```

Remember to create the required indexes first. You can also add a self-referential foreign key constraint to a table using ALTER TABLE.

InnoDB also supports the use of ALTER TABLE to drop foreign keys:

```

ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;

```

If the FOREIGN KEY clause included a CONSTRAINT name when you created the foreign key, you can refer to that name to drop the foreign key. Otherwise, the *fk_symbol* value is internally generated by InnoDB when the foreign key is created. To find out the symbol value when you want to drop a foreign key, use the SHOW CREATE TABLE statement. For example:

```

mysql> SHOW CREATE TABLE ibtest11c\G
***** 1. row *****
      Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',

```

```

`B` varchar(200) NOT NULL default '',
`C` varchar(175) default NULL,
PRIMARY KEY (`A`,`D`,`B`),
KEY `B` (`B`,`C`),
KEY `C` (`C`),
CONSTRAINT `0_38775` FOREIGN KEY (`A`,`D`)
REFERENCES `ibtest11a` (`A`,`D`)
ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT `0_38776` FOREIGN KEY (`B`,`C`)
REFERENCES `ibtest11a` (`B`,`C`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB CHARSET=latin1
1 row in set (0.01 sec)

```

```
mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY `0_38775`;
```

You cannot add a foreign key and drop a foreign key in separate clauses of a single ALTER TABLE statement. Separate statements are required.

The InnoDB parser allows table and column identifiers in a FOREIGN KEY ... REFERENCES ... clause to be quoted within backticks. (Alternatively, double quotes can be used if the ANSI_QUOTES SQL mode is enabled.) The InnoDB parser also takes into account the setting of the lower_case_table_names system variable.

InnoDB returns a table's foreign key definitions as part of the output of the SHOW CREATE TABLE statement:

```
SHOW CREATE TABLE tbl_name;
```

mysqldump also produces correct definitions of tables to the dump file, and does not forget about the foreign keys.

You can also display the foreign key constraints for a table like this:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

The foreign key constraints are listed in the Comment column of the output.

When performing foreign key checks, InnoDB sets shared row-level locks on child or parent records it has to look at. InnoDB checks foreign key constraints immediately; the check is not deferred to transaction commit.

To make it easier to reload dump files for tables that have foreign key

relationships, **mysqldump** automatically includes a statement in the dump output to set `FOREIGN_KEY_CHECKS` to 0. This avoids problems with tables having to be reloaded in a particular order when the dump is reloaded. It is also possible to set this variable manually:

```
mysql> SET FOREIGN_KEY_CHECKS = 0;  
mysql> SOURCE dump_file_name;  
mysql> SET FOREIGN_KEY_CHECKS = 1;
```

This allows you to import the tables in any order if the dump file contains tables that are not correctly ordered for foreign keys. It also speeds up the import operation. Setting `FOREIGN_KEY_CHECKS` to 0 can also be useful for ignoring foreign key constraints during `LOAD DATA` and `ALTER TABLE` operations. However, even if `FOREIGN_KEY_CHECKS=0`, InnoDB does not allow the creation of a foreign key constraint where a column references a non-matching column type.

InnoDB does not allow you to drop a table that is referenced by a `FOREIGN KEY` constraint, unless you do `SET FOREIGN_KEY_CHECKS=0`. When you drop a table, the constraints that were defined in its create statement are also dropped.

If you re-create a table that was dropped, it must have a definition that conforms to the foreign key constraints referencing it. It must have the right column names and types, and it must have indexes on the referenced keys, as stated earlier. If these are not satisfied, MySQL returns error number 1005 and refers to `errno 150` in the error message.

14.2.6.5. InnoDB and MySQL Replication

MySQL replication works for InnoDB tables as it does for MyISAM tables. It is also possible to use replication in a way where the storage engine on the slave is not the same as the original storage engine on the master. For example, you can replicate modifications to an InnoDB table on the master to a MyISAM table on the slave.

To set up a new slave for a master, you have to make a copy of the InnoDB tablespace and the log files, as well as the `.frm` files of the InnoDB tables, and move the copies to the slave. If the `innodb_file_per_table` variable is enabled, you must also copy the `.ibd` files as well. For the proper procedure to do this, see [Section 14.2.8, “Backing Up and Recovering an InnoDB Database”](#).

If you can shut down the master or an existing slave, you can take a cold backup of the InnoDB tablespace and log files and use that to set up a slave. To make a new slave without taking down any server you can also use the non-free (commercial) [InnoDB Hot Backup tool](#).

You cannot set up replication for InnoDB using the `LOAD TABLE FROM MASTER` statement, which works only for MyISAM tables. There are two possible workarounds:

- Dump the table on the master and import the dump file into the slave.
- Use `ALTER TABLE tbl_name ENGINE=MyISAM` on the master before setting up replication with `LOAD TABLE tbl_name FROM MASTER`, and then use `ALTER TABLE` to convert the master table back to InnoDB afterward. However, this should not be done for tables that have foreign key definitions because the definitions will be lost.

Transactions that fail on the master do not affect replication at all. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A transaction that fails (for example, because of a foreign key violation, or because it is rolled back) is not written to the binary log, so it is not sent to slaves. See [Section 13.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

14.2.7. Adding and Removing InnoDB Data and Log Files

This section describes what you can do when your InnoDB tablespace runs out of room or when you want to change the size of the log files.

The easiest way to increase the size of the InnoDB tablespace is to configure it from the beginning to be auto-extending. Specify the `autoextend` attribute for the last data file in the tablespace definition. Then InnoDB increases the size of that file automatically in 8MB increments when it runs out of space. The increment size can be changed by setting the value of the `innodb_autoextend_increment` system variable, which is measured in MB.

Alternatively, you can increase the size of your tablespace by adding another data file. To do this, you have to shut down the MySQL server, change the tablespace configuration to add a new data file to the end of

`innodb_data_file_path`, and start the server again.

If your last data file was defined with the keyword `autoextend`, the procedure for reconfiguring the tablespace must take into account the size to which the last data file has grown. Obtain the size of the data file, round it down to the closest multiple of 1024×1024 bytes (= 1MB), and specify the rounded size explicitly in `innodb_data_file_path`. Then you can add another data file. Remember that only the last data file in the `innodb_data_file_path` can be specified as `auto-extending`.

As an example, assume that the tablespace has just one auto-extending data file `ibdata1`:

```
innodb_data_home_dir =  
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that this data file, over time, has grown to 988MB. Here is the configuration line after modifying the original data file to not be auto-extending and adding another auto-extending data file:

```
innodb_data_home_dir =  
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:auto
```

When you add a new file to the tablespace configuration, make sure that it does not exist. InnoDB will create and initialize the file when you restart the server.

Currently, you cannot remove a data file from the tablespace. To decrease the size of your tablespace, use this procedure:

1. Use **mysqldump** to dump all your InnoDB tables.
2. Stop the server.
3. Remove all the existing tablespace files.
4. Configure a new tablespace.
5. Restart the server.
6. Import the dump files.

If you want to change the number or the size of your InnoDB log files, use the following instructions. The procedure to use depends on the value of `innodb_fast_shutdown`:

- If `innodb_fast_shutdown` is not set to 2: You must stop the MySQL server and make sure that it shuts down without errors (to ensure that there is no information for outstanding transactions in the logs). Then copy the old log files into a safe place just in case something went wrong in the shutdown and you need them to recover the tablespace. Delete the old log files from the log file directory, edit `my.cnf` to change the log file configuration, and start the MySQL server again. **mysqld** sees that no log files exist at startup and tells you that it is creating new ones.
- If `innodb_fast_shutdown` is set to 2: You should shut down the server, set `innodb_fast_shutdown` to 1, and restart the server. The server should be allowed to recover. Then you should shut down the server again and follow the procedure described in the preceding item to change InnoDB log file size. Set `innodb_fast_shutdown` back to 2 and restart the server.

14.2.8. Backing Up and Recovering an InnoDB Database

The key to safe database management is making regular backups.

InnoDB Hot Backup is an online backup tool you can use to backup your InnoDB database while it is running. **InnoDB Hot Backup** does not require you to shut down your database and it does not set any locks or disturb your normal database processing. **InnoDB Hot Backup** is a non-free (commercial) add-on tool with an annual license fee of €390 per computer on which the MySQL server is run. See the [InnoDB Hot Backup home page](#) for detailed information and screenshots.

If you are able to shut down your MySQL server, you can make a binary backup that consists of all files used by InnoDB to manage its tables. Use the following procedure:

1. Shut down your MySQL server and make sure that it shuts down without errors.
2. Copy all your data files (ibdata files and `.ibd` files) into a safe place.

3. Copy all your `ib_logfile` files to a safe place.
4. Copy your `my.cnf` configuration file or files to a safe place.
5. Copy all the `.frm` files for your InnoDB tables to a safe place.

Replication works with InnoDB tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability.

In addition to making binary backups as just described, you should also regularly make dumps of your tables with **mysqldump**. The reason for this is that a binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, because the format is simpler, the chance for serious data corruption is smaller. **mysqldump** also has a `--single-transaction` option that you can use to make a consistent snapshot without locking out other clients.

To be able to recover your InnoDB database to the present from the binary backup just described, you have to run your MySQL server with binary logging turned on. Then you can apply the binary log to the backup database to achieve point-in-time recovery:

```
mysqlbinlog yourhostname-bin.123 | mysql
```

To recover from a crash of your MySQL server, the only requirement is to restart it. InnoDB automatically checks the logs and performs a roll-forward of the database to the present. InnoDB automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, **mysqld** displays output something like this:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
```

```
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

If your database gets corrupted or your disk fails, you have to do the recovery from a backup. In the case of corruption, you should first find a backup that is not corrupted. After restoring the base backup, do the recovery from the binary log files using **mysqlbinlog** and **mysql** to restore the changes performed after the backup was made.

In some cases of database corruption it is enough just to dump, drop, and re-create one or a few corrupt tables. You can use the `CHECK TABLE SQL` statement to check whether a table is corrupt, although `CHECK TABLE` naturally cannot detect every possible kind of corruption. You can use `innodb_tablespace_monitor` to check the integrity of the file space management inside the tablespace files.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best first to try restarting your computer. Doing so may eliminate errors that appeared to be database page corruption.

14.2.8.1. Forcing InnoDB Recovery

If there is database page corruption, you may want to dump your tables from the database with `SELECT INTO outfile`. Usually, most of the data obtained in this way is intact. Even so, the corruption may cause `SELECT * FROM tbl_name` statements or InnoDB background operations to crash or assert, or even to cause InnoDB roll-forward recovery to crash. However, you can force the InnoDB storage engine to start up while preventing background operations from running, so that you are able to dump your tables. For example, you can add the following line to the `[mysqld]` section of your option file before restarting the server:

```
[mysqld]
innodb_force_recovery = 4
```

The allowable non-zero values for `innodb_force_recovery` follow. A larger number includes all precautions of smaller numbers. If you are able to dump your tables with an option value of at most 4, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 6 is more drastic because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

- 1 (`SRV_FORCE_IGNORE_CORRUPT`)

Let the server run even if it detects a corrupt page. Try to make `SELECT * FROM tbl_name` jump over corrupt index records and pages, which helps in dumping tables.

- 2 (`SRV_FORCE_NO_BACKGROUND`)

Prevent the main thread from running. If a crash would occur during the purge operation, this recovery value prevents it.

- 3 (`SRV_FORCE_NO_TRX_UNDO`)

Do not run transaction rollbacks after recovery.

- 4 (`SRV_FORCE_NO_IBUF_MERGE`)

Prevent also insert buffer merge operations. If they would cause a crash, do not do them. Do not calculate table statistics.

- 5 (`SRV_FORCE_NO_UNDO_LOG_SCAN`)

Do not look at undo logs when starting the database: InnoDB treats even incomplete transactions as committed.

- 6 (`SRV_FORCE_NO_LOG_REDO`)

Do not do the log roll-forward in connection with recovery.

You can `SELECT` from tables to dump them, or `DROP` or `CREATE` tables even if forced recovery is used. If you know that a given table is causing a crash on rollback, you can drop it. You can also use this to stop a runaway rollback caused by a failing mass import or `ALTER TABLE`. You can kill the **mysqld**

process and set `innodb_force_recovery` to 3 to bring the database up without the rollback, then DROP the table that is causing the runaway rollback.

The database must not otherwise be used with any non-zero value of `innodb_force_recovery`. As a safety measure, InnoDB prevents users from performing INSERT, UPDATE, or DELETE operations when `innodb_force_recovery` is greater than 0.

14.2.8.2. Checkpoints

InnoDB implements a checkpoint mechanism known as “fuzzy” checkpointing. InnoDB flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would in practice stop processing of user SQL statements during the checkpointing process.

During crash recovery, InnoDB looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then InnoDB scans the log files forward from the checkpoint, applying the logged modifications to the database.

InnoDB writes to its log files on a rotating basis. All committed modifications that make the database pages in the buffer pool different from the images on disk must be available in the log files in case InnoDB has to do a recovery. This means that when InnoDB starts to reuse a log file, it has to make sure that the database page images on disk contain the modifications logged in the log file that InnoDB is going to reuse. In other words, InnoDB must create a checkpoint and this often involves flushing of modified database pages to disk.

The preceding description explains why making your log files very large may save disk I/O in checkpointing. It often makes sense to set the total size of the log files as big as the buffer pool or even bigger. The drawback of using large log files is that crash recovery can take longer because there is more logged information to apply to the database.

14.2.9. Moving an InnoDB Database to Another Machine

On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from

Windows to Unix, you should have all table and database names in lowercase. A convenient way to accomplish this is to add the following line to the `[mysqld]` section of your `my.cnf` or `my.ini` file before creating any databases or tables:

```
[mysqld]
lower_case_table_names=1
```

Like MyISAM data files, InnoDB data and log files are binary-compatible on all platforms having the same floating-point number format. You can move an InnoDB database simply by copying all the relevant files listed in [Section 14.2.8, “Backing Up and Recovering an InnoDB Database”](#). If the floating-point formats differ but you have not used `FLOAT` or `DOUBLE` data types in your tables, then the procedure is the same: simply copy the relevant files. If the formats differ and your tables contain floating-point data, you must use **mysqldump** to dump your tables on one machine and then import the dump files on the other machine.

One way to increase performance is to switch off autocommit mode when importing data, assuming that the tablespace has enough space for the big rollback segment that the import transactions generate. Do the commit only after importing a whole table or a segment of a table.

14.2.10. InnoDB Transaction Model and Locking

In the InnoDB transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking. InnoDB does locking on the row level and runs queries as non-locking consistent reads by default, in the style of Oracle. The lock table in InnoDB is stored so space-efficiently that lock escalation is not needed: Typically several users are allowed to lock every row in the database, or any random subset of the rows, without InnoDB running out of memory.

14.2.10.1. InnoDB Lock Modes

InnoDB implements standard row-level locking where there are two types of locks:

- A shared (*s*) lock allows a transaction to read a row (tuple).
- An exclusive (*x*) lock allows a transaction to update or delete a row.

If transaction τ_1 holds a shared (S) lock on tuple t , then

- A request from some distinct transaction τ_2 for an S lock on t can be granted immediately. As a result, both τ_1 and τ_2 hold an S lock on t .
- A request from some distinct transaction τ_2 for an X lock on t cannot be granted immediately.

If a transaction τ_1 holds an exclusive (X) lock on tuple t , then a request from some distinct transaction τ_2 for a lock of either type on t cannot be granted immediately. Instead, transaction τ_2 has to wait for transaction τ_1 to release its lock on tuple t .

Additionally, InnoDB supports *multiple granularity locking* which allows coexistence of record locks and locks on entire tables. To make locking at multiple granularity levels practical, additional types of locks called *intention locks* are used. Intention locks are table locks in InnoDB. The idea behind intention locks is for a transaction to indicate which type of lock (shared or exclusive) it will require later for a row in that table. There are two types of intention locks used in InnoDB (assume that transaction τ has requested a lock of the indicated type on table R):

- Intention shared (IS): Transaction τ intends to set S locks on individual rows in table R .
- Intention exclusive (IX): Transaction τ intends to set X locks on those rows.

The intention locking protocol is as follows:

- Before a given transaction can acquire an S lock on a given row, it must first acquire an IS or stronger lock on the table containing that row.
- Before a given transaction can acquire an X lock on a given row, it must first acquire an IX lock on the table containing that row.

These rules can be conveniently summarized by means of a *lock type compatibility matrix*:

	X	IX	S	IS
X	Conflict	Conflict	Conflict	Conflict

<i>IX</i>	Conflict	Compatible	Conflict	Compatible
<i>S</i>	Conflict	Conflict	Compatible	Compatible
<i>IS</i>	Conflict	Compatible	Compatible	Compatible

A lock is granted to a requesting transaction if it is compatible with existing locks. A lock is not granted to a requesting transaction if it conflicts with existing locks. A transaction waits until the conflicting existing lock is released. If a lock request conflicts with an existing lock and cannot be granted because it would cause deadlock, an error occurs.

Thus, intention locks do not block anything except full table requests (for example, `LOCK TABLES ... WRITE`). The main purpose of *IX* and *IS* locks is to show that someone is locking a row, or going to lock a row in the table.

The following example illustrates how an error can occur when a lock request would cause a deadlock. The example involves two clients, A and B.

First, client A creates a table containing one row, and then begins a transaction. Within the transaction, A obtains an *S* lock on the row by selecting it in share mode:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;
+-----+
| i     |
+-----+
| 1    |
+-----+
1 row in set (0.10 sec)
```

Next, client B begins a transaction and attempts to delete the row from the table:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DELETE FROM t WHERE i = 1;
```

The delete operation requires an x lock. The lock cannot be granted because it is incompatible with the s lock that client A holds, so the request goes on the queue of lock requests for the row and client B blocks.

Finally, client A also attempts to delete the row from the table:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

Deadlock occurs here because client A needs an x lock to delete the row. However, that lock request cannot be granted because client B is already has a request for an x lock and is waiting for client A to release its s lock. Nor can the s lock held by A be upgraded to an x lock because of the prior request by B for an x lock. As a result, InnoDB generates an error for client A and releases its locks. At that point, the lock request for client B can be granted and B deletes the row from the table.

14.2.10.2. InnoDB and AUTOCOMMIT

In InnoDB, all user activity occurs inside a transaction. If the autocommit mode is enabled, each SQL statement forms a single transaction on its own. By default, MySQL starts new connections with autocommit enabled.

If the autocommit mode is switched off with `SET AUTOCOMMIT = 0`, then we can consider that a user always has a transaction open. An SQL `COMMIT` or `ROLLBACK` statement ends the current transaction and a new one starts. A `COMMIT` means that the changes made in the current transaction are made permanent and become visible to other users. A `ROLLBACK` statement, on the other hand, cancels all modifications made by the current transaction. Both statements release all InnoDB locks that were set during the current transaction.

If the connection has autocommit enabled, the user can still perform a multiple-statement transaction by starting it with an explicit `START TRANSACTION` or `BEGIN` statement and ending it with `COMMIT` or `ROLLBACK`.

14.2.10.3. InnoDB and TRANSACTION ISOLATION LEVEL

In terms of the SQL:1992 transaction isolation levels, the InnoDB default is REPEATABLE READ. InnoDB offers all four transaction isolation levels described by the SQL standard. You can set the default isolation level for all connections by using the `--transaction-isolation` option on the command line or in an option file. For example, you can set the option in the `[mysqld]` section of an option file like this:

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                        | REPEATABLE-READ | SERIALIZABLE}
```

A user can change the isolation level for a single session or for all new incoming connections with the `SET TRANSACTION` statement. Its syntax is as follows:

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
    {READ UNCOMMITTED | READ COMMITTED
     | REPEATABLE READ | SERIALIZABLE}
```

Note that there are hyphens in the level names for the `--transaction-isolation` option, but not for the `SET TRANSACTION` statement.

The default behavior is to set the isolation level for the next (not started) transaction. If you use the `GLOBAL` keyword, the statement sets the default transaction level globally for all new connections created from that point on (but not for existing connections). You need the `SUPER` privilege to do this. Using the `SESSION` keyword sets the default transaction level for all future transactions performed on the current connection.

Any client is free to change the session isolation level (even in the middle of a transaction), or the isolation level for the next transaction.

You can determine the global and session transaction isolation levels by checking the value of the `tx_isolation` system variable with these statements:

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

In row-level locking, InnoDB uses next-key locking. That means that besides index records, InnoDB can also lock the “gap” preceding an index record to block insertions by other users immediately before the index record. A next-key lock refers to a lock that locks an index record and the gap before it. A gap lock refers to a lock that only locks a gap before some index record.

A detailed description of each isolation level in InnoDB follows:

- READ UNCOMMITTED

SELECT statements are performed in a non-locking fashion, but a possible earlier version of a record might be used. Thus, using this isolation level, such reads are not consistent. This is also called a “dirty read.” Otherwise, this isolation level works like READ COMMITTED.

- READ COMMITTED

A somewhat Oracle-like isolation level. All SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE statements lock only the index records, not the gaps before them, and thus allow the free insertion of new records next to locked records. UPDATE and DELETE statements using a unique index with a unique search condition lock only the index record found, not the gap before it. In range-type UPDATE and DELETE statements, InnoDB must set next-key or gap locks and block insertions by other users to the gaps covered by the range. This is necessary because “phantom rows” must be blocked for MySQL replication and recovery to work.

Consistent reads behave as in Oracle: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See [Section 14.2.10.4, “Consistent Non-Locking Read”](#).

- REPEATABLE READ

This is the default isolation level of InnoDB. SELECT ... FOR UPDATE, SELECT ... LOCK IN SHARE MODE, UPDATE, and DELETE statements that use a unique index with a unique search condition lock only the index record found, not the gap before it. With other search conditions, these operations employ next-key locking, locking the index range scanned with next-key or gap locks, and block new insertions by other users.

In consistent reads, there is an important difference from the READ COMMITTED isolation level: All consistent reads within the same transaction read the same snapshot established by the first read. This convention means that if you issue several plain SELECT statements within the same transaction, these SELECT statements are consistent also with respect to each other. See [Section 14.2.10.4, “Consistent Non-Locking Read”](#).

- SERIALIZABLE

This level is like REPEATABLE READ, but InnoDB implicitly commits all plain SELECT statements to SELECT ... LOCK IN SHARE MODE.

14.2.10.4. Consistent Non-Locking Read

A consistent read means that InnoDB uses multi-versioning to present to a query a snapshot of the database at a point in time. The query sees the changes made by those transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by earlier statements within the same transaction. Note that the exception to the rule causes the following anomaly: if you update some rows in a table, a SELECT will see the latest version of the updated rows, while it sees the old version of other rows. If other users simultaneously update the same table, the anomaly means that you may see the table in a state that never existed in the database.

If you are running with the default REPEATABLE READ isolation level, all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

Consistent read is the default mode in which InnoDB processes SELECT statements in READ COMMITTED and REPEATABLE READ isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other users are free to modify those tables at the same time a consistent read is being performed on the table.

Note that consistent read does not work over DROP TABLE and over ALTER TABLE. Consistent read does not work over DROP TABLE because MySQL can't use a table that has been dropped and InnoDB destroys the table. Consistent read does not work over ALTER TABLE because ALTER TABLE works by making a temporary copy of the original table and deleting the original table when the temporary copy is built. When you reissue a consistent read within a transaction, rows in the new table are not visible because those rows did not exist when the transaction's snapshot was taken.

14.2.10.5. SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE

Locking Reads

In some circumstances, a consistent read is not convenient. For example, you might want to add a new row into your table `child`, and make sure that the child has a parent in table `parent`. The following example shows how to implement referential integrity in your application code.

Suppose that you use a consistent read to read the table `parent` and indeed see the parent of the child in the table. Can you safely add the child row to table `child`? No, because it may happen that meanwhile some other user deletes the parent row from the table `parent` without you being aware of it.

The solution is to perform the `SELECT` in a locking mode using `LOCK IN SHARE MODE`:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

Performing a read in share mode means that we read the latest available data, and set a shared mode lock on the rows we read. A shared mode lock prevents others from updating or deleting the row we have read. Also, if the latest data belongs to a yet uncommitted transaction of another client connection, we wait until that transaction commits. After we see that the preceding query returns the parent 'Jones', we can safely add the child record to the `child` table and commit our transaction.

Let us look at another example: We have an integer counter field in a table `child_codes` that we use to assign a unique identifier to each child added to table `child`. Obviously, using a consistent read or a shared mode read to read the present value of the counter is not a good idea because two users of the database may then see the same value for the counter, and a duplicate-key error occurs if two users attempt to add children with the same identifier to the table.

Here, `LOCK IN SHARE MODE` is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when attempting to update the counter.

In this case, there are two good ways to implement the reading and incrementing of the counter: (1) update the counter first by incrementing it by 1 and only after that read it, or (2) read the counter first with a lock mode `FOR UPDATE`, and increment after that. The latter approach can be implemented as follows:

```
SELECT counter_field FROM child_codes FOR UPDATE;  
UPDATE child_codes SET counter_field = counter_field + 1;
```

A `SELECT ... FOR UPDATE` reads the latest available data, setting exclusive locks on each row it reads. Thus, it sets the same locks a searched SQL `UPDATE` would set on the rows.

The preceding description is merely an example of how `SELECT ... FOR UPDATE` works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field  
SELECT LAST_INSERT_ID());
```

The `SELECT` statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

Locks set by `IN SHARE MODE` and `FOR UPDATE` reads are released when the transaction is committed or rolled back.

14.2.10.6. Next-Key Locking: Avoiding the Phantom Problem

In row-level locking, InnoDB uses an algorithm called *next-key locking*. InnoDB performs the row-level locking in such a way that when it searches or scans an index of a table, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index record locks.

The locks InnoDB sets on index records also affect the “gap” before that index record. If a user has a shared or exclusive lock on record R in an index, another user cannot insert a new index record immediately before R in the index order. This locking of gaps is done to prevent the so-called “phantom problem.” Suppose that you want to read and lock all children from the `child` table having an identifier value greater than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

Suppose that there is an index on the `id` column. The query scans that index starting from the first record where `id` is bigger than 100. If the locks set on the index records would not lock out inserts made in the gaps, a new row might

meanwhile be inserted to the table. If you execute the same `SELECT` within the same transaction, you would see a new row in the result set returned by the query. This is contrary to the isolation principle of transactions: A transaction should be able to run so that the data it has read does not change during the transaction. If we regard a set of rows as a data item, the new “phantom” child would violate this isolation principle.

When InnoDB scans an index, it can also lock the gap after the last record in the index. Just that happens in the previous example: The locks set by InnoDB prevent any insert to the table where `id` would be bigger than 100.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus, the next-key locking allows you to “lock” the non-existence of something in your table.

14.2.10.7. An Example of Consistent Read in InnoDB

Suppose that you are running in the default `REPEATABLE READ` isolation level. When you issue a consistent read (that is, an ordinary `SELECT` statement), InnoDB gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.

You can advance your timepoint by committing your transaction and then doing another `SELECT`.

This is called *multi-versioned concurrency control*.

	User A	User B
time	<code>SET AUTOCOMMIT=0;</code>	<code>SET AUTOCOMMIT=0;</code>
	<code>SELECT * FROM t;</code>	
	empty set	
		<code>INSERT INTO t VALUES (1, 2);</code>
v	<code>SELECT * FROM t;</code>	

```

empty set
                                COMMIT;

SELECT * FROM t;
empty set

COMMIT;

SELECT * FROM t;
-----
|    1    |    2    |
-----
1 row in set

```

In this example, user A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

If you want to see the “freshest” state of the database, you should use either the READ COMMITTED isolation level or a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

14.2.10.8. Locks Set by Different SQL Statements in InnoDB

A locking read, an UPDATE, or a DELETE generally set record locks on every index record that is scanned in the processing of the SQL statement. It does not matter if there are WHERE conditions in the statement that would exclude the row. InnoDB does not remember the exact WHERE condition, but only knows which index ranges were scanned. The record locks are normally next-key locks that also block inserts to the “gap” immediately before the record.

If the locks to be set are exclusive, InnoDB always retrieves also the clustered index record and sets a lock on it.

If you do not have indexes suitable for your statement and MySQL has to scan the whole table to process the statement, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily need to scan many rows.

InnoDB sets specific types of locks as follows:

- `SELECT ... FROM` is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to `SERIALIZABLE`. For `SERIALIZABLE` level, this sets shared next-key locks on the index records it encounters.
- `SELECT ... FROM ... LOCK IN SHARE MODE` sets shared next-key locks on all index records the read encounters.
- `SELECT ... FROM ... FOR UPDATE` sets exclusive next-key locks on all index records the read encounters.
- `INSERT INTO ... VALUES (...)` sets an exclusive lock on the inserted row. Note that this lock is not a next-key lock and does not prevent other users from inserting to the gap before the inserted row. If a duplicate-key error occurs, a shared lock on the duplicate index record is set.
- While initializing a previously specified `AUTO_INCREMENT` column on a table, InnoDB sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column. In accessing the auto-increment counter, InnoDB uses a specific table lock mode `AUTO-INC` where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Note that other clients cannot insert into the table while the `AUTO-INC` table lock is held; see [Section 14.2.10.2, “InnoDB and AUTOCOMMIT”](#).

InnoDB fetches the value of a previously initialized `AUTO_INCREMENT` column without setting any locks.

- `INSERT INTO T SELECT ... FROM S WHERE ...` sets an exclusive (non-next-key) lock on each row inserted into `T`. InnoDB sets shared next-key locks on `S`, unless `innodb_locks_unsafe_for_binlog` is enabled, in which case it does the search on `S` as a consistent read. InnoDB has to set locks in the latter case: In roll-forward recovery from a backup, every SQL statement has to be executed in exactly the same way it was done originally.
- `CREATE TABLE ... SELECT ...` performs the `SELECT` as a consistent read or with shared locks, as in the previous item.
- `REPLACE` is done like an insert if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row that has to be

updated.

- `UPDATE ... WHERE ...` sets an exclusive next-key lock on every record the search encounters.
- `DELETE FROM ... WHERE ...` sets an exclusive next-key lock on every record the search encounters.
- If a `FOREIGN KEY` constraint is defined on a table, any insert, update, or delete that requires the constraint condition to be checked sets shared record-level locks on the records that it looks at to check the constraint. InnoDB also sets these locks in the case where the constraint fails.
- `LOCK TABLES` sets table locks, but it is the higher MySQL layer above the InnoDB layer that sets these locks. InnoDB is aware of table locks if `innodb_table_locks=1` (the default) and `AUTOCOMMIT=0`, and the MySQL layer above InnoDB knows about row-level locks. Otherwise, InnoDB's automatic deadlock detection cannot detect deadlocks where such table locks are involved. Also, because the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another user currently has row-level locks. However, this does not endanger transaction integrity, as discussed in [Section 14.2.10.10, “Deadlock Detection and Rollback”](#). See also [Section 14.2.16, “Restrictions on InnoDB Tables”](#).

14.2.10.9. Implicit Transaction Commit and Rollback

By default, MySQL begins each client connection with autocommit mode enabled. When autocommit is enabled, MySQL does a commit after each SQL statement if that statement did not return an error. If an SQL statement returns an error, the commit or rollback behavior depends on the error. See [Section 14.2.15, “InnoDB Error Handling”](#).

If you have the autocommit mode off and close a connection without explicitly committing the final transaction, MySQL rolls back that transaction.

Each of the following statements (and any synonyms for them) implicitly end a transaction, as if you had done a `COMMIT` before executing the statement:

- ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, BEGIN, CREATE DATABASE, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, DROP DATABASE, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, LOAD MASTER DATA, LOCK TABLES, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE, UNLOCK TABLES.
- UNLOCK TABLES commits a transaction only if any tables are currently locked.
- The CREATE TABLE, CREATE DATABASE DROP DATABASE, and TRUNCATE TABLE statements cause an implicit commit beginning with MySQL 5.0.8. The ALTER FUNCTION, ALTER PROCEDURE, CREATE FUNCTION, CREATE PROCEDURE, DROP FUNCTION, and DROP PROCEDURE statements cause an implicit commit beginning with MySQL MySQL 5.0.13.
- The CREATE TABLE statement in InnoDB is processed as a single transaction. This means that a ROLLBACK from the user does not undo CREATE TABLE statements the user made during that transaction.

Transactions cannot be nested. This is a consequence of the implicit COMMIT performed for any current transaction when you issue a START TRANSACTION statement or one of its synonyms.

Statements that cause implicit cannot be used in an XA transaction while the transaction is in an ACTIVE state.

14.2.10.10. Deadlock Detection and Rollback

InnoDB automatically detects a deadlock of transactions and rolls back a transaction or transactions to break the deadlock. InnoDB tries to pick small transactions to roll back, where the size of a transaction is determined by the number of rows inserted, updated, or deleted.

InnoDB is aware of table locks if innodb_table_locks=1 (the default) and AUTOCOMMIT=0, and the MySQL layer above it knows about row-level locks. Otherwise, InnoDB cannot detect deadlocks where a table lock set by a MySQL LOCK TABLES statement or a lock set by a storage engine other than InnoDB is involved. You must resolve these situations by setting the value of the innodb_lock_wait_timeout system variable.

When InnoDB performs a complete rollback of a transaction, all locks set by the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the statement may be preserved. This happens because InnoDB stores row locks in a format such that it cannot know afterward which lock was set by which statement.

14.2.10.11. How to Cope with Deadlocks

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really “atomic”; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- Use `SHOW ENGINE INNODB STATUS` to determine the cause of the latest deadlock. That can help you to tune your application to avoid deadlocks.
- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.
- Commit your transactions often. Small transactions are less prone to collision.
- If you are using locking reads (`SELECT ... FOR UPDATE` or `... LOCK IN SHARE MODE`), try using a lower isolation level such as `READ COMMITTED`.
- Access your tables and rows in a fixed order. Then transactions form well-defined queues and do not deadlock.
- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use `EXPLAIN SELECT` to determine which indexes the MySQL server regards as the most

appropriate for your queries.

- Use less locking. If you can afford to allow a `SELECT` to return data from an old snapshot, do not add the clause `FOR UPDATE` or `LOCK IN SHARE MODE` to it. Using the `READ COMMITTED` isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.
- If nothing else helps, serialize your transactions with table-level locks. The correct way to use `LOCK TABLES` with transactional tables, such as InnoDB tables, is to set `AUTOCOMMIT = 0` and not to call `UNLOCK TABLES` until after you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

```
SET AUTOCOMMIT=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

Table-level locks make your transactions queue nicely, and deadlocks are avoided.

- Another way to serialize transactions is to create an auxiliary “semaphore” table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the InnoDB instant deadlock detection algorithm also works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.
- In applications that use the `LOCK TABLES` command, MySQL does not set InnoDB table locks if `AUTOCOMMIT=1`.

14.2.11. InnoDB Performance Tuning Tips

- In InnoDB, having a long `PRIMARY KEY` wastes a lot of disk space because its value must be stored with every secondary index record. (See [Section 14.2.13, “InnoDB Table and Index Structures”](#).) Create an `AUTO_INCREMENT` column as the primary key if your primary key is long.

- If the Unix `top` tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound. Maybe you are making too many transaction commits, or the buffer pool is too small. Making the buffer pool bigger can help, but do not set it equal to more than 80% of physical memory.
- Wrap several modifications into one transaction. InnoDB must flush the log to disk at each transaction commit if that transaction made modifications to the database. The rotation speed of a disk is typically at most 167 revolutions/second, which constrains the number of commits to the same 167th of a second if the disk does not “fool” the operating system.
- If you can afford the loss of some of the latest committed transactions if a crash occurs, you can set the `innodb_flush_log_at_trx_commit` parameter to 0. InnoDB tries to flush the log once per second anyway, although the flush is not guaranteed.
- Make your log files big, even as big as the buffer pool. When InnoDB has written the log files full, it has to write the modified contents of the buffer pool to disk in a checkpoint. Small log files cause many unnecessary disk writes. The drawback of big log files is that the recovery time is longer.
- Make the log buffer quite large as well (on the order of 8MB).
- Use the `VARCHAR` data type instead of `CHAR` if you are storing variable-length strings or if the column may contain many `NULL` values. A `CHAR(N)` column always takes *N* characters to store data, even if the string is shorter or its value is `NULL`. Smaller tables fit better in the buffer pool and reduce disk I/O.

When using `row_format=compact` (the default InnoDB record format in MySQL 5.0) and variable-length character sets, such as `utf8` or `sjis`, `CHAR(N)` will occupy a variable amount of space, at least *N* bytes.

- In some versions of GNU/Linux and Unix, flushing files to disk with the Unix `fsync()` call (which InnoDB uses by default) and other similar methods is surprisingly slow. If you are dissatisfied with database write performance, you might try setting the `innodb_flush_method` parameter to `O_DSYNC`. Although `O_DSYNC` seems to be slower on most systems, yours

might not be one of them.

- When using the InnoDB storage engine on Solaris 10 for x86_64 architecture (AMD Opteron), it is important to mount any filesystems used for storing InnoDB-related files using the `forcedirectio` option. (The default on Solaris 10/x86_64 is *not* to use this option.) Failure to use `forcedirectio` causes a serious degradation of InnoDB's speed and performance on this platform.

When using the InnoDB storage engine with a large `innodb_buffer_pool_size` value on any release of Solaris 2.6 and up and any platform (sparc/x86/x64/amd64), a significant performance gain can be achieved by placing InnoDB data files and log files on raw devices or on a separate direct I/O UFS filesystem (using mount option `forcedirectio`; see `mount_ufs(1M)`). Users of the Veritas filesystem VxFS should use the mount option `convosync=direct`.

Other MySQL data files, such as those for MyISAM tables, should not be placed on a direct I/O filesystem. Executables or libraries *must not* be placed on a direct I/O filesystem.

- When importing data into InnoDB, make sure that MySQL does not have autocommit mode enabled because that requires a log flush to disk for every insert. To disable autocommit during your import operation, surround it with `SET AUTOCOMMIT` and `COMMIT` statements:

```
SET AUTOCOMMIT=0;  
... SQL import statements ...  
COMMIT;
```

If you use the **`mysqldump`** option `--opt`, you get dump files that are fast to import into an InnoDB table, even without wrapping them with the `SET AUTOCOMMIT` and `COMMIT` statements.

- Beware of big rollbacks of mass inserts: InnoDB uses the insert buffer to save disk I/O in inserts, but no such mechanism is used in a corresponding rollback. A disk-bound rollback can take 30 times as long to perform as the corresponding insert. Killing the database process does not help because the rollback starts again on server startup. The only way to get rid of a runaway rollback is to increase the buffer pool so that the rollback becomes CPU-

bound and runs fast, or to use a special procedure. See [Section 14.2.8.1, “Forcing InnoDB Recovery”](#).

- Beware also of other big disk-bound operations. Use `DROP TABLE` and `CREATE TABLE` to empty a table, not `DELETE FROM tbl_name`.
- Use the multiple-row `INSERT` syntax to reduce communication overhead between the client and the server if you need to insert many rows:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

This tip is valid for inserts into any table, not just InnoDB tables.

- If you have `UNIQUE` constraints on secondary keys, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

```
SET UNIQUE_CHECKS=0;  
... import operation ...  
SET UNIQUE_CHECKS=1;
```

For big tables, this saves a lot of disk I/O because InnoDB can use its insert buffer to write secondary index records in a batch. Be certain that the data contains no duplicate keys. `UNIQUE_CHECKS` allows but does not require storage engines to ignore duplicate keys.

- If you have `FOREIGN KEY` constraints in your tables, you can speed up table imports by turning the foreign key checks off for the duration of the import session:

```
SET FOREIGN_KEY_CHECKS=0;  
... import operation ...  
SET FOREIGN_KEY_CHECKS=1;
```

For big tables, this can save a lot of disk I/O.

- If you often have recurring queries for tables that are not updated frequently, use the query cache:

```
[mysqld]  
query_cache_type = ON  
query_cache_size = 10M
```

14.2.11.1. SHOW ENGINE INNODB STATUS and the InnoDB Monitors

InnoDB includes InnoDB Monitors that print information about the InnoDB internal state. You can use the `SHOW ENGINE INNODB STATUS` SQL statement at any time to fetch the output of the standard InnoDB Monitor to your SQL client. This information is useful in performance tuning. (If you are using the **mysql** interactive SQL client, the output is more readable if you replace the usual semicolon statement terminator with `\G`.) For a discussion of InnoDB lock modes, see [Section 14.2.10.1, “InnoDB Lock Modes”](#).

```
mysql> SHOW ENGINE INNODB STATUS\G
```

Another way to use InnoDB Monitors is to let them periodically write data to the standard output of the **mysqld** server. In this case, no output is sent to clients. When switched on, InnoDB Monitors print data about every 15 seconds. Server output usually is directed to the `.err` log in the MySQL data directory. This data is useful in performance tuning. On Windows, you must start the server from a command prompt in a console window with the `--console` option if you want to direct the output to the window rather than to the error log.

Monitor output includes the following types of information:

- Table and record locks held by each active transaction
- Lock waits of a transactions
- Semaphore waits of threads
- Pending file I/O requests
- Buffer pool statistics
- Purge and insert buffer merge activity of the main InnoDB thread

To cause the standard InnoDB Monitor to write to the standard output of **mysqld**, use the following SQL statement:

```
CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
```

The monitor can be stopped by issuing the following statement:

```
DROP TABLE innodb_monitor;
```

The CREATE TABLE syntax is just a way to pass a command to the InnoDB engine through MySQL's SQL parser: The only things that matter are the table name `innodb_monitor` and that it be an InnoDB table. The structure of the table is not relevant at all for the InnoDB Monitor. If you shut down the server, the monitor does not restart automatically when you restart the server. You must drop the monitor table and issue a new CREATE TABLE statement to start the monitor. (This syntax may change in a future release.)

You can use `innodb_lock_monitor` in a similar fashion. This is the same as `innodb_monitor`, except that it also provides a great deal of lock information. A separate `innodb_tablespace_monitor` prints a list of created file segments existing in the tablespace and validates the tablespace allocation data structures. In addition, there is `innodb_table_monitor` with which you can print the contents of the InnoDB internal data dictionary.

A sample of InnoDB Monitor output:

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
Status:
=====
030709 13:00:59 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 18 seconds
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds th
semaphore: X-lock on RW-latch at 41a28668 created in file btr0sea.c
a writer (thread id 32782) has reserved it in mode wait exclusive
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
Last time write locked in file btr0sea.c line 1347
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 108462, OS waits 37964; RW-excl spins 681824, OS wai
375485
-----
LATEST FOREIGN KEY ERROR
-----
030709 13:00:59 Transaction:
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id
inserting
```

```

15 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'kHdK' , 'kHdK')
Foreign key constraint fails for table test/ibtest11a:
'
  CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b
  (`D`) ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index PRIMARY tuple:
  0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....
  len 4; hex 6b68446b; asc kHdK;; 3: len 6; hex 0000114e0edc; asc ...
  len 7; hex 00000000c3e0a7; asc .....;; 5: len 4; hex 6b68446b; as
But in parent table test/ibtest11b, in index PRIMARY,
the closest match we can find is record:
RECORD: info bits 0 0: len 4; hex 8000015b; asc ...[;; 1: len 4; hex
80000005; asc ....;; 2: len 3; hex 6b6864; asc khd;; 3: len 6; hex
0000111ef3eb; asc .....;; 4: len 7; hex 800001001e0084; asc .....
len 3; hex 6b6864; asc khd;;
-----
LATEST DETECTED DEADLOCK
-----
030709 12:59:58
*** (1) TRANSACTION:
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185, OS thread id
inserting
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146
MySQL thread id 21, query id 4553379 localhost heikki update
INSERT INTO alex1 VALUES(86, 86, 794, 'aA35818', 'bb', 'c79166', 'd4766t
'e187358f', 'g84586', 'h794', date_format('2001-04-03 12:54:22', '%Y-%m-
%H:%i'), 7
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 in
symbole trx id 0 290252780 lock mode S waiting
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 616133353
asc aa35818;; 1:
*** (2) TRANSACTION:
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190, OS thread id
inserting
130 lock struct(s), heap size 11584, undo log entries 437
MySQL thread id 23, query id 4554396 localhost heikki update
REPLACE INTO alex1 VALUES(NULL, 32, NULL, 'aa3572', '', 'c3572', 'd6012t
NULL, 'h396', NULL, NULL, 7.31,7.31,7.31,200)
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 in
symbole trx id 0 290251546 lock_mode X locks rec but not gap
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 616133353
asc aa35818;; 1:
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 in
symbole trx id 0 290251546 lock_mode X locks gap before rec insert i
waiting

```

```

Record lock, heap no 82 RECORD: info bits 0 0: len 7; hex 6161333537
asc aa35720;; 1:
*** WE ROLL BACK TRANSACTION (1)
-----
TRANSACTIONS
-----
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
Total number of lock structs in row lock hash table 70
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3491, OS thread id 42002
MySQL thread id 32, query id 4668737 localhost heikki
show innodb status
---TRANSACTION 0 290328384, ACTIVE 0 sec, process no 3205, OS thread
38929 inserting
1 lock struct(s), heap size 320
MySQL thread id 29, query id 4668736 localhost heikki update
insert into speedc values (1519229,1, 'hgjhjgghggjggjggjggjggjggjggj
jllhhggghggggghjhghggggghjhghghghghghhhghghghjhjhghjghjkgjhghjghjgh
---TRANSACTION 0 290328383, ACTIVE 0 sec, process no 3180, OS thread
28684 committing
1 lock struct(s), heap size 320, undo log entries 1
MySQL thread id 19, query id 4668734 localhost heikki update
insert into speedcm values (1603393,1, 'hgjhjgghggjggjggjggjggjggjggj
gjlhhggghggggghjhghggggghjhghghghghghhhghghghjhjhghjghjkgjhghjghjg
---TRANSACTION 0 290328327, ACTIVE 0 sec, process no 3200, OS thread
36880 starting index read
LOCK WAIT 2 lock struct(s), heap size 320
MySQL thread id 27, query id 4668644 localhost heikki Searching rows
update
update ibtest11a set B = 'kHdkkkk' where A = 89572
----- TRX HAS BEEN WAITING 0 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 65556 n bits 232 table test/ibtest11
PRIMARY trx id 0 290328327 lock_mode X waiting
Record lock, heap no 1 RECORD: info bits 0 0: len 9; hex 73757072656
asc supremum.;;
-----
---TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread
34831 rollback of SQL statement
ROLLING BACK 14 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' , 'khDk')
---TRANSACTION 0 290327208, ACTIVE 1 sec, process no 3190, OS thread
32782
58 lock struct(s), heap size 5504, undo log entries 159
MySQL thread id 23, query id 4668732 localhost heikki update
REPLACE INTO alex1 VALUES(86, 46, 538, 'aa95666', 'bb', 'c95666', 'd9486
'e200498f', 'g86814', 'h538', date_format('2001-04-03 12:54:22', '%Y-%m-
%H:%i'),
---TRANSACTION 0 290323325, ACTIVE 3 sec, process no 3185, OS thread

```

30733 inserting
4 lock struct(s), heap size 1024, undo log entries 165
MySQL thread id 21, query id 4668735 localhost heikki update
INSERT INTO alex1 VALUES(NULL, 49, NULL, 'aa42837', '', 'c56319', 'd1719
NULL, 'h321', NULL, NULL, 7.31, 7.31, 7.31, 200)

FILE I/O

I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
151671 OS file reads, 94747 OS file writes, 8750 OS fsyncs
25.44 reads/s, 18494 avg bytes/read, 17.55 writes/s, 2.33 fsyncs/s

INSERT BUFFER AND ADAPTIVE HASH INDEX

Ibuf for space 0: size 1, free list len 19, seg size 21,
85004 inserts, 85004 merged recs, 26669 merges
Hash table size 207619, used cells 14461, node heap has 16 buffer(s)
1877.67 hash searches/s, 5121.10 non-hash searches/s

LOG

Log sequence number 18 1212842764
Log flushed up to 18 1212665295
Last checkpoint at 18 1135877290
0 pending log writes, 0 pending chkp writes
4341 log i/o's done, 1.22 log i/o's/second

BUFFER POOL AND MEMORY

Total memory allocated 84966343; in additional pool allocated 140262
Buffer pool size 3200
Free buffers 110
Database pages 3074
Modified db pages 2674
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 171380, created 51968, written 194688
28.72 reads/s, 20.72 creates/s, 47.55 writes/s
Buffer pool hit rate 999 / 1000

ROW OPERATIONS

0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging

```
Number of rows inserted 3738558, updated 127415, deleted 33707, read
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
-----
END OF INNODB MONITOR OUTPUT
=====
```

Some notes on the output:

- If the TRANSACTIONS section reports lock waits, your applications may have lock contention. The output can also help to trace the reasons for transaction deadlocks.
- The SEMAPHORES section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside InnoDB. Contention can be due to heavy parallelism of queries or problems in operating system thread scheduling. Setting `innodb_thread_concurrency` smaller than the default value can help in such situations.
- The BUFFER POOL AND MEMORY section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.
- The ROW OPERATIONS section shows what the main thread is doing.

InnoDB sends diagnostic output to `stderr` or to files rather than to `stdout` or fixed-size memory buffers, to avoid potential buffer overflows. As a side effect, the output of `SHOW ENGINE INNODB STATUS` is written to a status file in the MySQL data directory every fifteen seconds. The name of the file is `innodb_status.pid`, where *pid* is the server process ID. InnoDB removes the file for a normal shutdown. If abnormal shutdowns have occurred, instances of these status files may be present and must be removed manually. Before removing them, you might want to examine them to see whether they contain useful information about the cause of abnormal shutdowns. The `innodb_status.pid` file is created only if the configuration option `innodb_status_file=1` is set.

14.2.12. Implementation of Multi-Versioning

Because InnoDB is a multi-versioned storage engine, it must keep information about old versions of rows in the tablespace. This information is stored in a data structure called a *rollback segment* (after an analogous data structure in Oracle).

Internally, InnoDB adds two fields to each row stored in the database. A 6-byte field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated.

InnoDB uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, but they can be discarded only after there is no transaction present for which InnoDB has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

You must remember to commit your transactions regularly, including those transactions that issue only consistent reads. Otherwise, InnoDB cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space need for your rollback segment.

In the InnoDB multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. Only when InnoDB can discard the update undo log record written for the deletion can it also physically remove the corresponding row and its index records from the database. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

In a scenario where the user inserts and deletes rows in smallish batches at about

the same rate in the table, it is possible that the purge thread starts to lag behind, and the table grows bigger and bigger, making everything disk-bound and very slow. Even if the table carries just 10MB of useful data, it may grow to occupy 10GB with all the “dead” rows. In such a case, it would be good to throttle new row operations, and allocate more resources to the purge thread. The `innodb_max_purge_lag` system variable exists for exactly this purpose. See [Section 14.2.4, “InnoDB Startup Options and System Variables”](#), for more information.

14.2.13. InnoDB Table and Index Structures

MySQL stores its data dictionary information for tables in `.frm` files in database directories. This is true for all MySQL storage engines. But every InnoDB table also has its own entry in the InnoDB internal data dictionary inside the tablespace. When MySQL drops a table or a database, it has to delete both an `.frm` file or files, and the corresponding entries inside the InnoDB data dictionary. This is the reason why you cannot move InnoDB tables between databases simply by moving the `.frm` files.

Every InnoDB table has a special index called the *clustered index* where the data for the rows is stored. If you define a `PRIMARY KEY` on your table, the index of the primary key is the clustered index.

If you do not define a `PRIMARY KEY` for your table, MySQL picks the first `UNIQUE` index that has only `NOT NULL` columns as the primary key and InnoDB uses it as the clustered index. If there is no such index in the table, InnoDB internally generates a clustered index where the rows are ordered by the row ID that InnoDB assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in insertion order.

Accessing a row through the clustered index is fast because the row data is on the same page where the index search leads. If a table is large, the clustered index architecture often saves a disk I/O when compared to the traditional solution. (In many database systems, data storage uses a different page from the index record.)

In InnoDB, the records in non-clustered indexes (also called secondary indexes) contain the primary key value for the row. InnoDB uses this primary key value to

search for the row from the clustered index. Note that if the primary key is long, the secondary indexes use more space.

InnoDB compares CHAR and VARCHAR strings of different lengths such that the remaining length in the shorter string is treated as if padded with spaces.

14.2.13.1. Physical Structure of an Index

All InnoDB indexes are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB. When new records are inserted, InnoDB tries to leave 1/16 of the page free for future insertions and updates of the index records.

If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full. If the fill factor of an index page drops below 1/2, InnoDB tries to contract the index tree to free the page.

14.2.13.2. Insert Buffering

It is a common situation in database applications that the primary key is a unique identifier and new rows are inserted in the ascending order of the primary key. Thus, the insertions to the clustered index do not require random reads from a disk.

On the other hand, secondary indexes are usually non-unique, and insertions into secondary indexes happen in a relatively random order. This would cause a lot of random disk I/O operations without a special mechanism used in InnoDB.

If an index record should be inserted to a non-unique secondary index, InnoDB checks whether the secondary index page is in the buffer pool. If that is the case, InnoDB does the insertion directly to the index page. If the index page is not found in the buffer pool, InnoDB inserts the record to a special insert buffer structure. The insert buffer is kept so small that it fits entirely in the buffer pool, and insertions can be done very fast.

Periodically, the insert buffer is merged into the secondary index trees in the database. Often it is possible to merge several insertions to the same page of the index tree, saving disk I/O operations. It has been measured that the insert buffer

can speed up insertions into a table up to 15 times.

The insert buffer merging may continue to happen *after* the inserting transaction has been committed. In fact, it may continue to happen after a server shutdown and restart (see [Section 14.2.8.1, “Forcing InnoDB Recovery”](#)).

The insert buffer merging may take many hours, when many secondary indexes must be updated, and many rows have been inserted. During this time, disk I/O will be increased, which can cause significant slowdown on disk-bound queries. Another significant background I/O operation is the purge thread (see [Section 14.2.12, “Implementation of Multi-Versioning”](#)).

14.2.13.3. Adaptive Hash Indexes

If a table fits almost entirely in main memory, the fastest way to perform queries on it is to use hash indexes. InnoDB has a mechanism that monitors index searches made to the indexes defined for a table. If InnoDB notices that queries could benefit from building a hash index, it does so automatically.

Note that the hash index is always built based on an existing B-tree index on the table. InnoDB can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches that InnoDB observes for the B-tree index. A hash index can be partial: It is not required that the whole B-tree index is cached in the buffer pool. InnoDB builds hash indexes on demand for those pages of the index that are often accessed.

In a sense, InnoDB tailors itself through the adaptive hash index mechanism to ample main memory, coming closer to the architecture of main-memory databases.

14.2.13.4. Physical Row Structure

The physical record structure for InnoDB tables is dependent on the MySQL version and the optional `ROW_FORMAT` option used when the table was created. For InnoDB tables in MySQL earlier than 5.0.3, only the `REDUNDANT` row format was available. For MySQL 5.0.3 and later, the default is to use the `COMPACT` row format, but you can use the `REDUNDANT` format to retain compatibility with older versions of InnoDB tables.

Records in InnoDB ROW_FORMAT=REDUNDANT tables have the following characteristics:

- Each index record contains a six-byte header. The header is used to link together consecutive records, and also in row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a six-byte field for the transaction ID and a seven-byte field for the roll pointer.
- If no primary key was defined for a table, each clustered index record also contains a six-byte row ID field.
- Each secondary index record contains also all the fields defined for the clustered index key.
- A record contains also a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of these pointers is called the record directory. The area where these pointers point is called the data part of the record.
- Internally, InnoDB stores fixed-length character columns such as CHAR(10) in a fixed-length format. InnoDB truncates trailing spaces from VARCHAR columns.
- An SQL NULL value reserves 1 or 2 bytes in the record directory. Besides that, an SQL NULL value reserves zero bytes in the data part of the record if stored in a variable length column. In a fixed-length column, it reserves the fixed length of the column in the data part of the record. The motivation behind reserving the fixed space for NULL values is that it enables an update of the column from NULL to a non-NULL value to be done in place without causing fragmentation of the index page.

Records in InnoDB ROW_FORMAT=COMPACT tables have the following characteristics:

- Each index record contains a five-byte header that may be preceded by a variable-length header. The header is used to link together consecutive records, and also in row-level locking.

- The record header contains a bit vector for indicating NULL columns. The bit vector occupies $(n_nullable+7)/8$ bytes. Columns that are NULL will not occupy other space than the bit in this vector.
- For each non-NULL variable-length field, the record header contains the length of the column in one or two bytes. Two bytes will only be needed if part of the column is stored externally or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes.
- The record header is followed by the data contents of the columns. Columns that are NULL are omitted.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a six-byte field for the transaction ID and a seven-byte field for the roll pointer.
- If no primary key was defined for a table, each clustered index record also contains a six-byte row ID field.
- Each secondary index record contains also all the fields defined for the clustered index key.
- Internally, InnoDB stores fixed-length, fixed-width character columns such as CHAR(10) in a fixed-length format. InnoDB truncates trailing spaces from VARCHAR columns.
- Internally, InnoDB attempts to store UTF-8 CHAR(n) columns in n bytes by trimming trailing spaces. In ROW_FORMAT=REDUNDANT, such columns occupy 3*n bytes. The motivation behind reserving the minimum space n is that it in many cases enables an update of the column to be done in place without causing fragmentation of the index page.

14.2.14. InnoDB File Space Management and Disk I/O

14.2.14.1. InnoDB Disk I/O

InnoDB uses simulated asynchronous disk I/O: InnoDB creates a number of threads to take care of I/O operations, such as read-ahead.

There are two read-ahead heuristics in InnoDB:

- In sequential read-ahead, if InnoDB notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.
- In random read-ahead, if InnoDB notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

InnoDB uses a novel file flush technique called *doublewrite*. It adds safety to recovery following an operating system crash or a power outage, and improves performance on most varieties of Unix by reducing the need for `fsync()` operations.

Doublewrite means that before writing pages to a data file, InnoDB first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does InnoDB write the pages to their proper positions in the data file. If the operating system crashes in the middle of a page write, InnoDB can later find a good copy of the page from the doublewrite buffer during recovery.

14.2.14.2. File Space Management

The data files that you define in the configuration file form the tablespace of InnoDB. The files are simply concatenated to form the tablespace. There is no striping in use. Currently, you cannot define where within the tablespace your tables are allocated. However, in a newly created tablespace, InnoDB allocates space starting from the first data file.

The tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of 64 consecutive pages. The “files” inside a tablespace are called *segments* in InnoDB. The term “rollback segment” is somewhat confusing because it actually contains many tablespace segments.

Two segments are allocated for each index in InnoDB. One is for non-leaf nodes of the B-tree, the other is for the leaf nodes. The idea here is to achieve better sequentiality for the leaf nodes, which contain the data.

When a segment grows inside the tablespace, InnoDB allocates the first 32 pages to it individually. After that InnoDB starts to allocate whole extents to the segment. InnoDB can add to a large segment up to 4 extents at a time to ensure good sequentiality of data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an InnoDB tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a `SHOW TABLE STATUS` statement, InnoDB reports the extents that are definitely free in the tablespace. InnoDB always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, InnoDB contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only in an (automatic) purge operation after they are no longer needed for transaction rollbacks or consistent reads. (See [Section 14.2.12, “Implementation of Multi-Versioning”](#).)

14.2.14.3. Defragmenting a Table

If there are random insertions into or deletions from the indexes of a table, the indexes may become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

A symptom of fragmentation is that a table takes more space than it “should” take. How much that is exactly, is difficult to determine. All InnoDB data and indexes are stored in B-trees, and their fill factor may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it “should” take:

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

(In the preceding query, we are “fooling” the SQL optimizer into scanning the clustered index, rather than a secondary index.) Most disks can read 10 to 50MB/s, which can be used to estimate how fast a table scan should run.

It can speed up index scans if you periodically perform a “null” ALTER TABLE operation:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

That causes MySQL to rebuild the table. Another way to perform a defragmentation operation is to use **mysqldump** to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions to an index are always ascending and records are deleted only from the end, the InnoDB filesystem management algorithm guarantees that fragmentation in the index does not occur.

14.2.15. InnoDB Error Handling

Error handling in InnoDB is not always the same as specified in the SQL standard. According to the standard, any error during an SQL statement should cause the rollback of that statement. InnoDB sometimes rolls back only part of the statement, or the whole transaction. The following items describe how InnoDB performs error handling:

- If you run out of file space in the tablespace, a MySQL `Table is full` error occurs and InnoDB rolls back the SQL statement.
- A transaction deadlock causes InnoDB to roll back the entire transaction. In the case of a lock wait timeout, InnoDB also rolls back the entire transaction before MySQL 5.0.13; as of 5.0.13, InnoDB rolls back only the most recent SQL statement.

When a transaction rollback occurs due to a deadlock or lock wait timeout, it cancels the effect of the statements within the transaction. But if the start-transaction statement was `START TRANSACTION` or `BEGIN` statement, rollback does not cancel that statement. Further SQL statements become part of the transaction until the occurrence of `COMMIT`, `ROLLBACK`, or some SQL statement that causes an implicit commit.

- A duplicate-key error rolls back the SQL statement, if you have not specified the IGNORE option in your statement.
- A row too long error rolls back the SQL statement.
- Other errors are mostly detected by the MySQL layer of code (above the InnoDB storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During implicit rollbacks, as well as during the execution of an explicit ROLLBACK SQL command, SHOW PROCESSLIST displays Rolling back in the State column for the relevant connection.

14.2.15.1. InnoDB Error Codes

The following is a non-exhaustive list of common InnoDB-specific errors that you may encounter, with information about why each occurs and how to resolve the problem.

- 1005 (ER_CANT_CREATE_TABLE)

Cannot create table. If the error message refers to errno 150, table creation failed because a foreign key constraint was not correctly formed. If the error message refers to errno -1, table creation probably failed because the table included a column name that matched the name of an internal InnoDB table.

- 1016 (ER_CANT_OPEN_FILE)

Cannot find the InnoDB table from the InnoDB data files, although the .frm file for the table exists. See [Section 14.2.17.1, “Troubleshooting InnoDB Data Dictionary Operations”](#).

- 1114 (ER_RECORD_FILE_FULL)

InnoDB has run out of free space in the tablespace. You should reconfigure the tablespace to add a new data file.

- 1205 (ER_LOCK_WAIT_TIMEOUT)

Lock wait timeout expired. Transaction was rolled back.

- 1213 (ER_LOCK_DEADLOCK)

Transaction deadlock. You should rerun the transaction.

- 1216 (ER_NO_REFERENCED_ROW)

You are trying to add a row but there is no parent row, and a foreign key constraint fails. You should add the parent row first.

- 1217 (ER_ROW_IS_REFERENCED)

You are trying to delete a parent row that has children, and a foreign key constraint fails. You should delete the children first.

14.2.15.2. Operating System Error Codes

To print the meaning of an operating system error number, use the **perror** program that comes with the MySQL distribution.

The following table provides a list of some common Linux system error codes. For a more complete list, see [Linux source code](#).

- 1 (EPERM)

Operation not permitted

- 2 (ENOENT)

No such file or directory

- 3 (ESRCH)

No such process

- 4 (EINTR)

Interrupted system call

- 5 (EIO)
I/O error
- 6 (ENXIO)
No such device or address
- 7 (E2BIG)
Arg list too long
- 8 (ENOEXEC)
Exec format error
- 9 (EBADF)
Bad file number
- 10 (ECHILD)
No child processes
- 11 (EAGAIN)
Try again
- 12 (ENOMEM)
Out of memory
- 13 (EACCES)
Permission denied
- 14 (EFAULT)
Bad address
- 15 (ENOTBLK)

Block device required

- 16 (EBUSY)

Device or resource busy

- 17 (EEXIST)

File exists

- 18 (EXDEV)

Cross-device link

- 19 (ENODEV)

No such device

- 20 (ENOTDIR)

Not a directory

- 21 (EISDIR)

Is a directory

- 22 (EINVAL)

Invalid argument

- 23 (ENFILE)

File table overflow

- 24 (EMFILE)

Too many open files

- 25 (ENOTTY)

Inappropriate ioctl for device

- 26 (ETXTBSY)
Text file busy
- 27 (EFBIG)
File too large
- 28 (ENOSPC)
No space left on device
- 29 (ESPIPE)
Illegal seek
- 30 (EROFS)
Read-only file system
- 31 (EMLINK)
Too many links

The following table provides a list of some common Windows system error codes. For a complete list see the [Microsoft Web site](#).

- 1 (ERROR_INVALID_FUNCTION)
Incorrect function.
- 2 (ERROR_FILE_NOT_FOUND)
The system cannot find the file specified.
- 3 (ERROR_PATH_NOT_FOUND)
The system cannot find the path specified.
- 4 (ERROR_TOO_MANY_OPEN_FILES)

The system cannot open the file.

- 5 (ERROR_ACCESS_DENIED)

Access is denied.

- 6 (ERROR_INVALID_HANDLE)

The handle is invalid.

- 7 (ERROR_ARENA_TRASHED)

The storage control blocks were destroyed.

- 8 (ERROR_NOT_ENOUGH_MEMORY)

Not enough storage is available to process this command.

- 9 (ERROR_INVALID_BLOCK)

The storage control block address is invalid.

- 10 (ERROR_BAD_ENVIRONMENT)

The environment is incorrect.

- 11 (ERROR_BAD_FORMAT)

An attempt was made to load a program with an incorrect format.

- 12 (ERROR_INVALID_ACCESS)

The access code is invalid.

- 13 (ERROR_INVALID_DATA)

The data is invalid.

- 14 (ERROR_OUTOFMEMORY)

Not enough storage is available to complete this operation.

- 15 (ERROR_INVALID_DRIVE)

The system cannot find the drive specified.

- 16 (ERROR_CURRENT_DIRECTORY)

The directory cannot be removed.

- 17 (ERROR_NOT_SAME_DEVICE)

The system cannot move the file to a different disk drive.

- 18 (ERROR_NO_MORE_FILES)

There are no more files.

- 19 (ERROR_WRITE_PROTECT)

The media is write protected.

- 20 (ERROR_BAD_UNIT)

The system cannot find the device specified.

- 21 (ERROR_NOT_READY)

The device is not ready.

- 22 (ERROR_BAD_COMMAND)

The device does not recognize the command.

- 23 (ERROR_CRC)

Data error (cyclic redundancy check).

- 24 (ERROR_BAD_LENGTH)

The program issued a command but the command length is incorrect.

- 25 (ERROR_SEEK)

The drive cannot locate a specific area or track on the disk.

- 26 (ERROR_NOT_DOS_DISK)

The specified disk or diskette cannot be accessed.

- 27 (ERROR_SECTOR_NOT_FOUND)

The drive cannot find the sector requested.

- 28 (ERROR_OUT_OF_PAPER)

The printer is out of paper.

- 29 (ERROR_WRITE_FAULT)

The system cannot write to the specified device.

- 30 (ERROR_READ_FAULT)

The system cannot read from the specified device.

- 31 (ERROR_GEN_FAILURE)

A device attached to the system is not functioning.

- 32 (ERROR_SHARING_VIOLATION)

The process cannot access the file because it is being used by another process.

- 33 (ERROR_LOCK_VIOLATION)

The process cannot access the file because another process has locked a portion of the file.

- 34 (ERROR_WRONG_DISK)

The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.

- 36 (ERROR_SHARING_BUFFER_EXCEEDED)

Too many files opened for sharing.

- 38 (ERROR_HANDLE_EOF)

Reached the end of the file.

- 39 (ERROR_HANDLE_DISK_FULL)

The disk is full.

- 87 (ERROR_INVALID_PARAMETER)

The parameter is incorrect. (If this error occurs on Windows and you have enabled `innodb_file_per_table` in a server option file, add the line `innodb_flush_method=unbuffered` to the file as well.)

- 112 (ERROR_DISK_FULL)

The disk is full.

- 123 (ERROR_INVALID_NAME)

The filename, directory name, or volume label syntax is incorrect.

- 1450 (ERROR_NO_SYSTEM_RESOURCES)

Insufficient system resources exist to complete the requested service.

14.2.16. Restrictions on InnoDB Tables

- **Warning:** Do *not* convert MySQL system tables in the `mysql` database from MyISAM to InnoDB tables! This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or re-generate them with the `mysql_install_db` script.
- A table cannot contain more than 1000 columns.
- The internal maximum key length is 3500 bytes, but MySQL itself restricts

this to 1024 bytes.

- The maximum row length, except for VARCHAR, BLOB and TEXT columns, is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including also BLOB and TEXT columns, must be less than 4GB. InnoDB stores the first 768 bytes of a VARCHAR, BLOB, or TEXT column in the row, and the rest into separate pages.
- Although InnoDB supports row sizes larger than 65535 internally, you cannot define a row containing VARCHAR columns with a combined size larger than 65535:

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),  
-> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),  
-> f VARCHAR(10000), g VARCHAR(10000)) ENGINE=InnoDB;  
ERROR 1118 (42000): Row size too large. The maximum row size for  
used table type, not counting BLOBs, is 65535. You have to change  
columns to TEXT or BLOBs
```

- On some older operating systems, files must be less than 2GB. This is not a limitation of InnoDB itself, but if you require a large tablespace, you will need to configure it using several smaller data files rather than one or a file large data files.
- The combined size of the InnoDB log files must be less than 4GB.
- The minimum tablespace size is 10MB. The maximum tablespace size is four billion database pages (64TB). This is also the maximum size for a table.
- InnoDB tables do not support FULLTEXT indexes.
- InnoDB tables do not support spatial data types before MySQL 5.0.16.
- ANALYZE TABLE determines index cardinality (as displayed in the Cardinality column of SHOW INDEX output) by doing ten random dives to each of the index trees and updating index cardinality estimates accordingly. Note that because these are only estimates, repeated runs of ANALYZE TABLE may produce different numbers. This makes ANALYZE TABLE fast on InnoDB tables but not 100% accurate as it doesn't take all

rows into account.

MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you can try using `ANALYZE TABLE`. In the few cases that `ANALYZE TABLE` doesn't produce values good enough for your particular tables, you can use `FORCE INDEX` with your queries to force the use of a particular index, or set the `max_seeks_for_key` system variable to ensure that MySQL prefers index lookups over table scans. See [Section 5.2.2, “Server System Variables”](#), and [Section A.6, “Optimizer-Related Issues”](#).

- `SHOW TABLE STATUS` does not give accurate statistics on InnoDB tables, except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.
- InnoDB does not keep an internal count of rows in a table. (In practice, this would be somewhat complicated due to multi-versioning.) To process a `SELECT COUNT(*) FROM t` statement, InnoDB must scan an index of the table, which takes some time if the index is not entirely in the buffer pool. To get a fast count, you have to use a counter table you create yourself and let your application update it according to the inserts and deletes it does. If your table does not change often, using the MySQL query cache is a good solution. `SHOW TABLE STATUS` also can be used if an approximate row count is sufficient. See [Section 14.2.11, “InnoDB Performance Tuning Tips”](#).
- On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in binary format from Unix to Windows or from Windows to Unix, you should always use explicitly lowercase names when creating databases and tables.
- For an `AUTO_INCREMENT` column, you must always define an index for the table, and that index must contain just the `AUTO_INCREMENT` column. In MyISAM tables, the `AUTO_INCREMENT` column may be part of a multi-column index.
- In MySQL 5.0 before MySQL 5.0.3, InnoDB does not support the `AUTO_INCREMENT` table option for setting the initial sequence value in a `CREATE TABLE` or `ALTER TABLE` statement. To set the value with InnoDB, insert a dummy row with a value one less and delete that dummy row, or

insert the first row with an explicit value specified.

- While initializing a previously specified `AUTO_INCREMENT` column on a table, InnoDB sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column. In accessing the auto-increment counter, InnoDB uses a specific table lock mode `AUTO-INC` where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Note that other clients cannot insert into the table while the `AUTO-INC` table lock is held; see [Section 14.2.10.2, “InnoDB and AUTOCOMMIT”](#).
- When you restart the MySQL server, InnoDB may reuse an old value that was generated for an `AUTO_INCREMENT` column but never stored (that is, a value that was generated during an old transaction that was rolled back).
- When an `AUTO_INCREMENT` column runs out of values, InnoDB wraps a `BIGINT` to `-9223372036854775808` and `BIGINT UNSIGNED` to `1`. However, `BIGINT` values have 64 bits, so do note that if you were to insert one million rows per second, it would still take nearly three hundred thousand years before `BIGINT` reached its upper bound. With all other integer type columns, a duplicate-key error results. This is similar to how `MyISAM` works, because it is mostly general MySQL behavior and not about any storage engine in particular.
- `DELETE FROM tbl_name` does not regenerate the table but instead deletes all rows, one by one.
- Under some conditions, `TRUNCATE tbl_name` for an InnoDB table is mapped to `DELETE FROM tbl_name` and doesn't reset the `AUTO_INCREMENT` counter. See [Section 13.2.9, “TRUNCATE Syntax”](#).
- In MySQL 5.0, the MySQL `LOCK TABLES` operation acquires two locks on each table if `innodb_table_locks=1` (the default). In addition to a table lock on the MySQL layer, it also acquires an InnoDB table lock. Older versions of MySQL did not acquire InnoDB table locks; the old behavior can be selected by setting `innodb_table_locks=0`. If no InnoDB table lock is acquired, `LOCK TABLES` completes even if some records of the tables are being locked by other transactions.
- All InnoDB locks held by a transaction are released when the transaction is

committed or aborted. Thus, it does not make much sense to invoke `LOCK TABLES` on InnoDB tables in `AUTOCOMMIT=1` mode, because the acquired InnoDB table locks would be released immediately.

- Sometimes it would be useful to lock further tables in the course of a transaction. Unfortunately, `LOCK TABLES` in MySQL performs an implicit `COMMIT` and `UNLOCK TABLES`. An InnoDB variant of `LOCK TABLES` has been planned that can be executed in the middle of a transaction.
- The `LOAD TABLE FROM MASTER` statement for setting up replication slave servers does not work for InnoDB tables. A workaround is to alter the table to MyISAM on the master, do then the load, and after that alter the master table back to InnoDB. Do not do this if the tables use InnoDB-specific features such as foreign keys.
- The default database page size in InnoDB is 16KB. By recompiling the code, you can set it to values ranging from 8KB to 64KB. You must update the values of `UNIV_PAGE_SIZE` and `UNIV_PAGE_SIZE_SHIFT` in the `univ.i` source file.
- Currently, triggers are not activated by cascaded foreign key actions.
- You cannot create a table with a column name that matches the name of an internal InnoDB column (including `DB_ROW_ID`, `DB_TRX_ID`, `DB_ROLL_PTR` and `DB_MIX_ID`). In versions of MySQL before 5.0.21 this would cause a crash, since 5.0.21 the server will report error 1005 and refers to `errno -1` in the error message.
- As of MySQL 5.0.19, InnoDB does not ignore trailing spaces when comparing `BINARY` or `VARBINARY` column values. See [Section 11.4.2, “The BINARY and VARBINARY Types”](#) and [Section D.1.8, “Changes in release 5.0.19 \(04 March 2006\)”](#).

14.2.17. InnoDB Troubleshooting

The following general guidelines apply to troubleshooting InnoDB problems:

- When an operation fails or you suspect a bug, you should look at the MySQL server error log, which is the file in the data directory that has a

suffix of `.err`.

- When troubleshooting, it is usually best to run the MySQL server from the command prompt, rather than through the `mysqld_safe` wrapper or as a Windows service. You can then see what `mysqld` prints to the console, and so have a better grasp of what is going on. On Windows, you must start the server with the `--console` option to direct the output to the console window.
- Use the InnoDB Monitors to obtain information about a problem (see [Section 14.2.11.1, “SHOW ENGINE INNODB STATUS and the InnoDB Monitors”](#)). If the problem is performance-related, or your server appears to be hung, you should use `innodb_monitor` to print information about the internal state of InnoDB. If the problem is with locks, use `innodb_lock_monitor`. If the problem is in creation of tables or other data dictionary operations, use `innodb_table_monitor` to print the contents of the InnoDB internal data dictionary.
- If you suspect that a table is corrupt, run `CHECK TABLE` on that table.

14.2.17.1. Troubleshooting InnoDB Data Dictionary Operations

A specific issue with tables is that the MySQL server keeps data dictionary information in `.frm` files it stores in the database directories, whereas InnoDB also stores the information into its own data dictionary inside the tablespace files. If you move `.frm` files around, or if the server crashes in the middle of a data dictionary operation, the locations of the `.frm` files may end up out of synchrony with the locations recorded in the InnoDB internal data dictionary.

A symptom of an out-of-sync data dictionary is that a `CREATE TABLE` statement fails. If this occurs, you should look in the server's error log. If the log says that the table already exists inside the InnoDB internal data dictionary, you have an orphaned table inside the InnoDB tablespace files that has no corresponding `.frm` file. The error message looks like this:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
```

```
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

You can drop the orphaned table by following the instructions given in the error message. If you are still unable to use `DROP TABLE` successfully, the problem may be due to name completion in the **mysql** client. To work around this problem, start the **mysql** client with the `--skip-auto-rehash` option and try `DROP TABLE` again. (With name completion on, **mysql** tries to construct a list of table names, which fails when a problem such as just described exists.)

Another symptom of an out-of-sync data dictionary is that MySQL prints an error that it cannot open a `.InnoDB` file:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

In the error log you can find a message like this:

```
InnoDB: Cannot find table test/child2 from the internal data diction
InnoDB: of InnoDB though the .frm file for the table exists. Maybe y
InnoDB: have deleted and recreated InnoDB data files but have forgot
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

This means that there is an orphaned `.frm` file without a corresponding table inside InnoDB. You can drop the orphaned `.frm` file by deleting it manually.

If MySQL crashes in the middle of an `ALTER TABLE` operation, you may end up with an orphaned temporary table inside the InnoDB tablespace. Using `innodb_table_monitor` you can see listed a table whose name is `#sql- . . .`. You can perform SQL statements on tables whose name contains the character `#` if you enclose the name within backticks. Thus, you can drop such an orphaned table like any other orphaned table using the method described earlier. Note that to copy or rename a file in the Unix shell, you need to put the file name in double quotes if the file name contains `#`.

14.3. The MERGE Storage Engine

The MERGE storage engine, also known as the MRG_MyISAM engine, is a collection of identical MyISAM tables that can be used as one. “Identical” means that all tables have identical column and index information. You cannot merge MyISAM tables in which the columns are listed in a different order, do not have exactly the same columns, or have the indexes in different order. However, any or all of the MyISAM tables can be compressed with **myisampack**. See [Section 8.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”](#). Differences in table options such as AVG_ROW_LENGTH, MAX_ROWS, or PACK_KEYS do not matter.

When you create a MERGE table, MySQL creates two files on disk. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format, and an `.MRG` file contains the names of the tables that should be used as one. The tables do not have to be in the same database as the MERGE table itself.

You can use SELECT, DELETE, UPDATE, and INSERT on MERGE tables. You must have SELECT, UPDATE, and DELETE privileges on the MyISAM tables that you map to a MERGE table.

Note: The use of MERGE tables entails the following security issue: If a user has access to MyISAM table *t*, that user can create a MERGE table *m* that accesses *t*. However, if the user's privileges on *t* are subsequently revoked, the user can continue to access *t* by doing so through *m*. If this behavior is undesirable, you can start the server with the new `--skip-merge` option to disable the MERGE storage engine. This option is available as of MySQL 5.0.24.

If you DROP the MERGE table, you are dropping only the MERGE specification. The underlying tables are not affected.

To create a MERGE table, you must specify a `UNION=(list-of-tables)` clause that indicates which MyISAM tables you want to use as one. You can optionally specify an `INSERT_METHOD` option if you want inserts for the MERGE table to take place in the first or last table of the UNION list. Use a value of FIRST or LAST to cause inserts to be made in the first or last table, respectively. If you do not specify an `INSERT_METHOD` option or if you specify it with a value of NO, attempts to insert

rows into the MERGE table result in an error.

The following example shows how to create a MERGE table:

```
mysql> CREATE TABLE t1 (  
->   a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
->   message CHAR(20)) ENGINE=MyISAM;  
mysql> CREATE TABLE t2 (  
->   a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
->   message CHAR(20)) ENGINE=MyISAM;  
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');  
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');  
mysql> CREATE TABLE total (  
->   a INT NOT NULL AUTO_INCREMENT,  
->   message CHAR(20), INDEX(a))  
->   ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

The older term TYPE is supported as a synonym for ENGINE for backward compatibility, but ENGINE is the preferred term and TYPE is deprecated.

Note that the a column is indexed as a PRIMARY KEY in the underlying MyISAM tables, but not in the MERGE table. There it is indexed but not as a PRIMARY KEY because a MERGE table cannot enforce uniqueness over the set of underlying tables.

After creating the MERGE table, you can issue queries that operate on the group of tables as a whole:

```
mysql> SELECT * FROM total;  
+----+-----+  
| a | message |  
+----+-----+  
| 1 | Testing |  
| 2 | table   |  
| 3 | t1      |  
| 1 | Testing |  
| 2 | table   |  
| 3 | t2      |  
+----+-----+
```

To remap a MERGE table to a different collection of MyISAM tables, you can use one of the following methods:

- DROP the MERGE table and re-create it.

- Use `ALTER TABLE tbl_name UNION=(...)` to change the list of underlying tables.

MERGE tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with **myisampack**, and then create a MERGE table to use them as one.
- Obtain more speed. You can split a big read-only table based on some criteria, and then put individual tables on different disks. A MERGE table on this could be much faster than using the big table.
- Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the split tables for some queries and use a MERGE table for others. You can even have many different MERGE tables that use overlapping sets of tables.
- Perform more efficient repairs. It is easier to repair individual tables that are mapped to a MERGE table than to repair a single large table.
- Instantly map many tables as one. A MERGE table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, MERGE table collections are *very* fast to create or remap. (Note that you must still specify the index definitions when you create a MERGE table, even though no indexes are created.)
- If you have a set of tables from which you create a large table on demand, you should instead create a MERGE table on them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each MyISAM table is bound by this limit, but a collection of MyISAM tables is not.
- You can create an alias or synonym for a MyISAM table by defining a MERGE table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and `memcpy()` calls for each read).

The disadvantages of MERGE tables are:

- You can use only identical MyISAM tables for a MERGE table.
- You cannot use a number of MyISAM features in MERGE tables. For example, you cannot create FULLTEXT indexes on MERGE tables. (You can, of course, create FULLTEXT indexes on the underlying MyISAM tables, but you cannot search the MERGE table with a full-text search.)
- If the MERGE table is non-temporary, all underlying MyISAM tables must be non-temporary, too. If the MERGE table is temporary, the MyISAM tables can be any mix of temporary and non-temporary.
- MERGE tables use more file descriptors. If 10 clients are using a MERGE table that maps to 10 tables, the server uses $(10 \times 10) + 10$ file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Key reads are slower. When you read a key, the MERGE storage engine needs to issue a read on all underlying tables to check which one most closely matches the given key. To read the next key, the MERGE storage engine needs to search the read buffers to find the next key. Only when one key buffer is used up does the storage engine need to read the next key block. This makes MERGE keys much slower on eq_ref searches, but not much slower on ref searches. See [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#), for more information about eq_ref and ref.

Additional resources

- A forum dedicated to the MERGE storage engine is available at <http://forums.mysql.com/list.php?93>.

14.3.1. MERGE Table Problems

The following are known problems with MERGE tables:

- If you use ALTER TABLE to change a MERGE table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying MyISAM tables are copied into the altered table, which then uses the specified storage engine.
- REPLACE does not work.

- You cannot use `DROP TABLE`, `ALTER TABLE`, `DELETE` without a `WHERE` clause, `REPAIR TABLE`, `TRUNCATE TABLE`, `OPTIMIZE TABLE`, or `ANALYZE TABLE` on any of the tables that are mapped into an open `MERGE` table. If you do so, the `MERGE` table may still refer to the original table, which yields unexpected results. The easiest way to work around this deficiency is to ensure that no `MERGE` tables remain open by issuing a `FLUSH TABLES` statement prior to performing any of those operations.
- `DROP TABLE` on a table that is in use by a `MERGE` table does not work on Windows because the `MERGE` storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not allow open files to be deleted, so you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table.
- A `MERGE` table cannot maintain uniqueness constraints over the entire table. When you perform an `INSERT`, the data goes into the first or last `MyISAM` table (depending on the value of the `INSERT_METHOD` option). MySQL ensures that unique key values remain unique within that `MyISAM` table, but not across all the tables in the collection.
- When you create or alter `MERGE` table, there is no check to ensure that the underlying tables are existing `MyISAM` tables and have identical structures. When the `MERGE` table is used, MySQL checks that the row length for all mapped tables is equal, but this is not foolproof. If you create a `MERGE` table from dissimilar `MyISAM` tables, you are very likely to run into strange problems.

Similarly, if you create a `MERGE` table from non-`MyISAM` tables, or if you drop an underlying table or alter it to be a non-`MyISAM` table, no error for the `MERGE` table occurs until later when you attempt to use it.

- The order of indexes in the `MERGE` table and its underlying tables should be the same. If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table, and then use `ALTER TABLE` to add a non-unique index on the `MERGE` table, the index ordering is different for the tables if there was already a non-unique index in the underlying table. (This happens because `ALTER TABLE` puts `UNIQUE` indexes before non-unique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.

- If you encounter an error message similar to ERROR 1017 (HY000): Can't find file: 'mm.MRG' (errno: 2) it generally indicates that some of the base tables are not using the MyISAM storage engine. Confirm that all tables are MyISAM.
- There is a limit of 2^{32} ($\sim 4.295E+09$) rows to a MERGE table, just as there is with a MyISAM, it is therefore not possible to merge multiple MyISAM tables that exceed this limitation. However, you build MySQL with the `--with-big-tables` option then the row limitation is increased to $(2^{32})^2$ ($1.844E+19$) rows. See [Section 2.9.2, “Typical configure Options”](#). Beginning with MySQL 5.0.4 all standard binaries are built with this option.

14.4. The MEMORY (HEAP) Storage Engine

The MEMORY storage engine creates tables with contents that are stored in memory. Formerly, these were known as HEAP tables. MEMORY is the preferred term, although HEAP remains supported for backward compatibility.

Each MEMORY table is associated with one disk file. The filename begins with the table name and has an extension of `.frm` to indicate that it stores the table definition.

To specify explicitly that you want to create a MEMORY table, indicate that with an `ENGINE` table option:

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

The older term `TYPE` is supported as a synonym for `ENGINE` for backward compatibility, but `ENGINE` is the preferred term and `TYPE` is deprecated.

As indicated by the name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in `.frm` files on disk, but they are empty when the server restarts.

This example shows how you might create, use, and remove a MEMORY table:

```
mysql> CREATE TABLE test ENGINE=MEMORY
->     SELECT ip,SUM(downloads) AS down
->     FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MEMORY tables have the following characteristics:

- Space for MEMORY tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows are put in a linked list and are reused when you insert new data into the table. MEMORY tables also have none of the problems commonly associated with deletes plus inserts in

hashed tables.

- MEMORY tables can have up to 32 indexes per table, 16 columns per index and a maximum key length of 500 bytes.
- The MEMORY storage engine implements both HASH and BTREE indexes. You can specify one or the other for a given index by adding a USING clause as shown here:

```
CREATE TABLE lookup
  (id INT, INDEX USING HASH (id))
  ENGINE = MEMORY;
CREATE TABLE lookup
  (id INT, INDEX USING BTREE (id))
  ENGINE = MEMORY;
```

General characteristics of B-tree and hash indexes are described in [Section 7.4.5, “How MySQL Uses Indexes”](#).

- You can have non-unique keys in a MEMORY table. (This is an uncommon feature for implementations of hash indexes.)
- If you have a hash index on a MEMORY table that has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a BTREE index to avoid this problem.
- Columns that are indexed can contain NULL values.
- MEMORY tables use a fixed-length row storage format.
- MEMORY tables cannot contain BLOB or TEXT columns.
- MEMORY includes support for AUTO_INCREMENT columns.
- You can use INSERT DELAYED with MEMORY tables. See [Section 13.2.4.2, “INSERT DELAYED Syntax”](#).
- MEMORY tables are shared among all clients (just like any other non-TEMPORARY table).

- MEMORY table contents are stored in memory, which is a property that MEMORY tables share with internal tables that the server creates on the fly while processing queries. However, the two types of tables differ in that MEMORY tables are not subject to storage conversion, whereas internal tables are:
 - If an internal table becomes too large, the server automatically converts it to an on-disk table. The size limit is determined by the value of the `tmp_table_size` system variable.
 - MEMORY tables are never converted to disk tables. To ensure that you don't accidentally do anything foolish, you can set the `max_heap_table_size` system variable to impose a maximum size on MEMORY tables. For individual tables, you can also specify a `MAX_ROWS` table option in the `CREATE TABLE` statement.
- The server needs sufficient memory to maintain all MEMORY tables that are in use at the same time.
- To free memory used by a MEMORY table when you no longer require its contents, you should execute `DELETE` or `TRUNCATE TABLE`, or remove the table altogether using `DROP TABLE`.
- If you want to populate a MEMORY table when the MySQL server starts, you can use the `--init-file` option. For example, you can put statements such as `INSERT INTO ... SELECT` or `LOAD DATA INFILE` into this file to load the table from a persistent data source. See [Section 5.2.1, “mysqld Command Options”](#), and [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).
- If you are using replication, the master server's MEMORY tables become empty when it is shut down and restarted. However, a slave is not aware that these tables have become empty, so it returns out-of-date content if you select data from them. When a MEMORY table is used on the master for the first time since the master was started, a `DELETE` statement is written to the master's binary log automatically, thus synchronizing the slave to the master again. Note that even with this strategy, the slave still has outdated data in the table during the interval between the master's restart and its first use of the table. However, if you use the `--init-file` option to populate the MEMORY table on the master at startup, it ensures that this time interval is zero.

- The memory needed for one row in a MEMORY table is calculated using the following expression:

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) × 4)  
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) × 2)  
+ ALIGN(length_of_row+1, sizeof(char*))
```

ALIGN() represents a round-up factor to cause the row length to be an exact multiple of the char pointer size. sizeof(char*) is 4 on 32-bit machines and 8 on 64-bit machines.

Additional resources

- A forum dedicated to the MEMORY storage engine is available at <http://forums.mysql.com/list.php?92>.

14.5. The BDB (BerkeleyDB) Storage Engine

Sleepycat Software has provided MySQL with the Berkeley DB transactional storage engine. This storage engine typically is called BDB for short. BDB tables may have a greater chance of surviving crashes and are also capable of COMMIT and ROLLBACK operations on transactions.

Support for the BDB storage engine is included in MySQL source distributions is activated in MySQL-Max binary distributions. The MySQL source distribution comes with a BDB distribution that is patched to make it work with MySQL. You cannot use a non-patched version of BDB with MySQL.

We at MySQL AB work in close cooperation with Sleepycat to keep the quality of the MySQL/BDB interface high. (Even though Berkeley DB is in itself very tested and reliable, the MySQL interface is still considered gamma quality. We continue to improve and optimize it.)

When it comes to support for any problems involving BDB tables, we are committed to helping our users locate the problem and create reproducible test cases. Any such test case is forwarded to Sleepycat, which in turn helps us find and fix the problem. As this is a two-stage operation, any problems with BDB tables may take a little longer for us to fix than for other storage engines. However, we anticipate no significant difficulties with this procedure because the Berkeley DB code itself is used in many applications other than MySQL.

For general information about Berkeley DB, please visit the Sleepycat Web site, <http://www.sleepycat.com/>.

14.5.1. Operating Systems Supported by BDB

Currently, we know that the BDB storage engine works with the following operating systems:

- Linux 2.x Intel
- Sun Solaris (SPARC and x86)
- FreeBSD 4.x/5.x (x86, sparc64)

- IBM AIX 4.3.x
- SCO OpenServer
- SCO UnixWare 7.1.x
- Windows NT/2000/XP

The BDB storage engine does *not* work with the following operating systems:

- Linux 2.x Alpha
- Linux 2.x AMD64
- Linux 2.x IA-64
- Linux 2.x s390
- Mac OS X

Note: The preceding lists are not complete. We update them as we receive more information.

If you build MySQL from source with support for BDB tables, but the following error occurs when you start **mysqld**, it means that the BDB storage engine is not supported for your architecture:

```
bdb: architecture lacks fast mutexes: applications cannot be threaded  
Can't init databases
```

In this case, you must rebuild MySQL without BDB support or start the server with the `--skip-bdb` option.

14.5.2. Installing BDB

If you have downloaded a binary version of MySQL that includes support for Berkeley DB, simply follow the usual binary distribution installation instructions. (MySQL-Max distributions include BDB support.)

If you build MySQL from source, you can enable BDB support by invoking **configure** with the `--with-berkeley-db` option in addition to any other options

that you normally use. Download a MySQL 5.0 distribution, change location into its top-level directory, and run this command:

```
shell> ./configure --with-berkeley-db [other-options]
```

For more information, see [Section 5.3, “The `mysqld-max` Extended MySQL Server”](#), [Section 2.8, “Installing MySQL on Other Unix-Like Systems”](#), and [Section 2.9, “MySQL Installation Using a Source Distribution”](#).

14.5.3. BDB Startup Options

The following options to `mysqld` can be used to change the behavior of the BDB storage engine. For more information, see [Section 5.2.1, “`mysqld` Command Options”](#).

- `--bdb-home=path`

The base directory for BDB tables. This should be the same directory that you use for `--datadir`.

- `--bdb-lock-detect=method`

The BDB lock detection method. The option value should be `DEFAULT`, `OLDEST`, `RANDOM`, or `YOUNGEST`.

- `--bdb-logdir=file_name`

The BDB log file directory.

- `--bdb-no-recover`

Do not start Berkeley DB in recover mode.

- `--bdb-no-sync`

Don't synchronously flush the BDB logs. This option is deprecated; use `--skip-sync-bdb-logs` instead (see the description for `--sync-bdb-logs`).

- `--bdb-shared-data`

Start Berkeley DB in multi-process mode. (Do not use `DB_PRIVATE` when

initializing Berkeley DB.)

- `--bdb-tmpdir=path`

The BDB temporary file directory.

- `--skip-bdb`

Disable the BDB storage engine.

- `--sync-bdb-logs`

Synchronously flush the BDB logs. This option is enabled by default. Use `--skip-sync-bdb-logs` to disable it.

If you use the `--skip-bdb` option, MySQL does not initialize the Berkeley DB library and this saves a lot of memory. However, if you use this option, you cannot use BDB tables. If you try to create a BDB table, MySQL uses the default storage engine instead.

Normally, you should start **mysqld** without the `--bdb-no-recover` option if you intend to use BDB tables. However, this may cause problems when you try to start **mysqld** if the BDB log files are corrupted. See [Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”](#).

With the `bdb_max_lock` variable, you can specify the maximum number of locks that can be active on a BDB table. The default is 10,000. You should increase this if errors such as the following occur when you perform long transactions or when **mysqld** has to examine many rows to execute a query:

```
bdb: Lock table is out of available locks
Got error 12 from ...
```

You may also want to change the `binlog_cache_size` and `max_binlog_cache_size` variables if you are using large multiple-statement transactions. See [Section 5.12.3, “The Binary Log”](#).

See also [Section 5.2.2, “Server System Variables”](#).

14.5.4. Characteristics of BDB Tables

Each BDB table is stored on disk in two files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format, and a `.db` file contains the table data and indexes.

To specify explicitly that you want a BDB table, indicate that with an `ENGINE` table option:

```
CREATE TABLE t (i INT) ENGINE = BDB;
```

The older term `TYPE` is supported as a synonym for `ENGINE` for backward compatibility, but `ENGINE` is the preferred term and `TYPE` is deprecated.

BerkeleyDB is a synonym for BDB in the `ENGINE` table option.

The BDB storage engine provides transactional tables. The way you use these tables depends on the autocommit mode:

- If you are running with autocommit enabled (which is the default), changes to BDB tables are committed immediately and cannot be rolled back.
- If you are running with autocommit disabled, changes do not become permanent until you execute a `COMMIT` statement. Instead of committing, you can execute `ROLLBACK` to forget the changes.

You can start a transaction with the `START TRANSACTION` or `BEGIN` statement to suspend autocommit, or with `SET AUTOCOMMIT=0` to disable autocommit explicitly.

For more information about transactions, see [Section 13.4.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”](#).

The BDB storage engine has the following characteristics:

- BDB tables can have up to 31 indexes per table, 16 columns per index, and a maximum key size of 1024 bytes.
- MySQL requires a primary key in each BDB table so that each row can be uniquely identified. If you don't create one explicitly by declaring a `PRIMARY KEY`, MySQL creates and maintains a hidden primary key for you. The hidden key has a length of five bytes and is incremented for each insert

attempt. This key does not appear in the output of `SHOW CREATE TABLE` or `DESCRIBE`.

- The primary key is faster than any other index, because it is stored together with the row data. The other indexes are stored as the key data plus the primary key, so it's important to keep the primary key as short as possible to save disk space and get better speed.

This behavior is similar to that of InnoDB, where shorter primary keys save space not only in the primary index but in secondary indexes as well.

- If all columns that you access in a BDB table are part of the same index or part of the primary key, MySQL can execute the query without having to access the actual row. In a MyISAM table, this can be done only if the columns are part of the same index.
- Sequential scanning is slower for BDB tables than for MyISAM tables because the data in BDB tables is stored in B-trees and not in a separate data file.
- Key values are not prefix- or suffix-compressed like key values in MyISAM tables. In other words, key information takes a little more space in BDB tables compared to MyISAM tables.
- There are often holes in the BDB table to allow you to insert new rows in the middle of the index tree. This makes BDB tables somewhat larger than MyISAM tables.
- `SELECT COUNT(*) FROM tbl_name` is slow for BDB tables, because no row count is maintained in the table.
- The optimizer needs to know the approximate number of rows in the table. MySQL solves this by counting inserts and maintaining this in a separate segment in each BDB table. If you don't issue a lot of `DELETE` or `ROLLBACK` statements, this number should be accurate enough for the MySQL optimizer. However, MySQL stores the number only on close, so it may be incorrect if the server terminates unexpectedly. It should not be fatal even if this number is not 100% correct. You can update the row count by using `ANALYZE TABLE` or `OPTIMIZE TABLE`. See [Section 13.5.2.1, “ANALYZE TABLE Syntax”](#), and [Section 13.5.2.5, “OPTIMIZE TABLE Syntax”](#).

- Internal locking in BDB tables is done at the page level.
- `LOCK TABLES` works on BDB tables as with other tables. If you do not use `LOCK TABLES`, MySQL issues an internal multiple-write lock on the table (a lock that does not block other writers) to ensure that the table is properly locked if another thread issues a table lock.
- To support transaction rollback, the BDB storage engine maintains log files. For maximum performance, you can use the `--bdb-logdir` option to place the BDB logs on a different disk than the one where your databases are located.
- MySQL performs a checkpoint each time a new BDB log file is started, and removes any BDB log files that are not needed for current transactions. You can also use `FLUSH LOGS` at any time to checkpoint the Berkeley DB tables.

For disaster recovery, you should use table backups plus MySQL's binary log. See [Section 5.10.1, “Database Backups”](#).

Warning: If you delete old log files that are still in use, BDB is not able to do recovery at all and you may lose data if something goes wrong.

- Applications must always be prepared to handle cases where any change of a BDB table may cause an automatic rollback and any read may fail with a deadlock error.
- If you get a full disk with a BDB table, you get an error (probably error 28) and the transaction should roll back. This contrasts with MyISAM tables, for which **mysqld** waits for sufficient free disk space before continuing.

14.5.5. Restrictions on BDB Tables

The following list indicates restrictions that you must observe when using BDB tables:

- Each BDB table stores in its `.db` file the path to the file as it was created. This is done to enable detection of locks in a multi-user environment that supports symlinks. As a consequence of this, it is not possible to move BDB table files from one database directory to another.

- When making backups of BDB tables, you must either use **mysqldump** or else make a backup that includes the files for each BDB table (the `.frm` and `.db` files) as well as the BDB log files. The BDB storage engine stores unfinished transactions in its log files and requires them to be present when **mysqld** starts. The BDB logs are the files in the data directory with names of the form `log.NNNNNNNNNN` (ten digits).
- If a column that allows NULL values has a unique index, only a single NULL value is allowed. This differs from other storage engines, which allow multiple NULL values in unique indexes.

14.5.6. Errors That May Occur When Using BDB Tables

- If the following error occurs when you start **mysqld** after upgrading, it means that the current version of BDB doesn't support the old log file format:

```
bdb: Ignoring log file: .../log.NNNNNNNNNN:
unsupported log version #
```

In this case, you must delete all BDB logs from your data directory (the files that have names of the form `log.NNNNNNNNNN`) and restart **mysqld**. We also recommend that you then use **mysqldump --opt** to dump your BDB tables, drop the tables, and restore them from the dump file.

- If autocommit mode is disabled and you drop a BDB table that is referenced in another transaction, you may get error messages of the following form in your MySQL error log:

```
001119 23:43:56 bdb: Missing log fileid entry
001119 23:43:56 bdb: txn_abort: Log undo failed for LSN:
1 3644744: Invalid
```

This is not fatal, but the fix is not trivial. Until the problem is fixed, we recommend that you not drop BDB tables except while autocommit mode is enabled.

14.6. The EXAMPLE Storage Engine

The EXAMPLE storage engine is a stub engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

The EXAMPLE storage engine is included in MySQL-Max binary distributions. To enable this storage engine if you build MySQL from source, invoke **configure** with the `--with-example-storage-engine` option.

To examine the source for the EXAMPLE engine, look in the `sql/examples` directory of a MySQL source distribution.

When you create an EXAMPLE table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. No other files are created. No data can be stored into the table. Retrievals return an empty result.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;  
Query OK, 0 rows affected (0.78 sec)
```

```
mysql> INSERT INTO test VALUES(1),(2),(3);  
ERROR 1031 (HY000): Table storage engine for 'test' doesn't have thi
```

```
mysql> SELECT * FROM test;  
Empty set (0.31 sec)
```

The EXAMPLE storage engine does not support indexing.

14.7. The FEDERATED Storage Engine

The FEDERATED storage engine is available beginning with MySQL 5.0.3. It is a storage engine that accesses data in tables of remote databases rather than in local tables.

The FEDERATED storage engine is included in MySQL-Max binary distributions. To enable this storage engine if you build MySQL from source, invoke **configure** with the `--with-federated-storage-engine` option.

To examine the source for the FEDERATED engine, look in the `sql` directory of a source distribution for MySQL 5.0.3 or newer.

Additional resources

- A forum dedicated to the FEDERATED storage engine is available at <http://forums.mysql.com/list.php?105>.

14.7.1. Description of the FEDERATED Storage Engine

When you create a FEDERATED table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. No other files are created, because the actual data is in a remote table. This differs from the way that storage engines for local tables work.

For local database tables, data files are local. For example, if you create a MyISAM table named `users`, the MyISAM handler creates a data file named `users.MYD`. A handler for local tables reads, inserts, deletes, and updates data in local data files, and rows are stored in a format particular to the handler. To read rows, the handler must parse data into columns. To write rows, column values must be converted to the row format used by the handler and written to the local data file.

With the MySQL FEDERATED storage engine, there are no local data files for a table (for example, there is no `.MYD` file). Instead, a remote database stores the data that normally would be in the table. The local server connects to a remote server, and uses the MySQL client API to read, delete, update, and insert data in the remote table. Data retrieval is initiated via a `SELECT * FROM tbl_name` SQL statement. To read the result, rows are fetched one at a time by using the

mysql_fetch_row() C API function, and then converting the columns in the SELECT result set to the format that the FEDERATED handler expects.

The flow of information is as follows:

1. SQL calls issued locally
2. MySQL handler API (data in handler format)
3. MySQL client API (data converted to SQL calls)
4. Remote database -> MySQL client API
5. Convert result sets (if any) to handler format
6. Handler API -> Result rows or rows-affected count to local

14.7.2. How to use FEDERATED Tables

The procedure for using FEDERATED tables is very simple. Normally, you have two servers running, either both on the same host or on different hosts. (It is possible for a FEDERATED table to use another table that is managed by the same server, although there is little point in doing so.)

First, you must have a table on the remote server that you want to access by using a FEDERATED table. Suppose that the remote table is in the federated database and is defined like this:

```
CREATE TABLE test_table (  
    id      INT(20) NOT NULL AUTO_INCREMENT,  
    name   VARCHAR(32) NOT NULL DEFAULT '',  
    other  INT(20) NOT NULL DEFAULT '0',  
    PRIMARY KEY (id),  
    INDEX name (name),  
    INDEX other_key (other)  
)  
ENGINE=MyISAM  
DEFAULT CHARSET=latin1;
```

The example uses a MyISAM table, but the table could use any storage engine.

Next, create a FEDERATED table on the local server for accessing the remote table:

```

CREATE TABLE federated_table (
    id      INT(20) NOT NULL AUTO_INCREMENT,
    name    VARCHAR(32) NOT NULL DEFAULT '',
    other   INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://root@remote_host:9306/federated/test_table';

```

(Before MySQL 5.0.13, use COMMENT rather than CONNECTION.)

The structure of this table must be exactly the same as that of the remote table, except that the ENGINE table option should be FEDERATED and the CONNECTION table option is a connection string that indicates to the FEDERATED engine how to connect to the remote server.

The FEDERATED engine creates only the test_table.frm file in the federated database.

The remote host information indicates the remote server to which your local server connects, and the database and table information indicates which remote table to use as the data source. In this example, the remote server is indicated to be running as remote_host on port 9306, so there must be a MySQL server running on the remote host and listening to port 9306.

The general form of the connection string in the CONNECTION option is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Only mysql is supported as the *scheme* value at this point; the password and port number are optional.

Here are some example connection strings:

```

CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'

```

The use of CONNECTION for specifying the connection string is non-optimal and is

likely to change in future. Keep this in mind for applications that use FEDERATED tables. Such applications are likely to need modification if the format for specifying connection information changes.

Because any password given in the connection string is stored as plain text, it can be seen by any user who can use `SHOW CREATE TABLE` or `SHOW TABLE STATUS` for the FEDERATED table, or query the TABLES table in the INFORMATION_SCHEMA database.

14.7.3. Limitations of the FEDERATED Storage Engine

The following items indicate features that the FEDERATED storage engine does and does not support:

- In the first version, the remote server must be a MySQL server. Support by FEDERATED for other database engines may be added in the future.
- The remote table that a FEDERATED table points to *must* exist before you try to access the table through the FEDERATED table.
- It is possible for one FEDERATED table to point to another, but you must be careful not to create a loop.
- There is no support for transactions.
- There is no way for the FEDERATED engine to know if the remote table has changed. The reason for this is that this table must work like a data file that would never be written to by anything other than the database. The integrity of the data in the local table could be breached if there was any change to the remote database.
- The FEDERATED storage engine supports `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and indexes. It does not support `ALTER TABLE`, or any Data Definition Language statements other than `DROP TABLE`. The current implementation does not use Prepared statements.
- Any `DROP TABLE` statement issued against a FEDERATED table will only drop the local table, not the remote table.
- The implementation uses `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but not

HANDLER.

- FEDERATED tables do not work with the query cache.

Some of these limitations may be lifted in future versions of the FEDERATED handler.

14.8. The ARCHIVE Storage Engine

The ARCHIVE storage engine is used for storing large amounts of data without indexes in a very small footprint.

The ARCHIVE storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke **configure** with the `--with-archive-storage-engine` option.

To examine the source for the ARCHIVE engine, look in the `sql` directory of a MySQL source distribution.

You can check whether the ARCHIVE storage engine is available with this statement:

```
mysql> SHOW VARIABLES LIKE 'have_archive';
```

When you create an ARCHIVE table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine creates other files, all having names beginning with the table name. The data and metadata files have extensions of `.ARZ` and `.ARM`, respectively. An `.ARN` file may appear during optimization operations.

The ARCHIVE engine supports `INSERT` and `SELECT`, but not `DELETE`, `REPLACE`, or `UPDATE`. It does support `ORDER BY` operations, `BLOB` columns, and basically all but spatial data types (see [Section 16.4.1, “MySQL Spatial Data Types”](#)). The ARCHIVE engine uses row-level locking.

Storage: Rows are compressed as they are inserted. The ARCHIVE engine uses `zlib` lossless data compression (see <http://www.zlib.net/>). You can use `OPTIMIZE TABLE` to analyze the table and pack it into a smaller format (for a reason to use `OPTIMIZE TABLE`, see later in this section). Beginning with MySQL 5.0.15, the engine also supports `CHECK TABLE`. There are several types of insertions that are used:

- An `INSERT` statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A `SELECT` forces a flush to occur, unless the only insertions that have

come in were INSERT DELAYED (those flush as necessary). See [Section 13.2.4.2, “INSERT DELAYED Syntax”](#).

- A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A SELECT never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

Retrieval: On retrieval, rows are uncompressed on demand; there is no row cache. A SELECT operation performs a complete table scan: When a SELECT occurs, it finds out how many rows are currently available and reads that number of rows. SELECT is performed as a consistent read. Note that lots of SELECT statements during insertion can deteriorate the compression, unless only bulk or delayed inserts are used. To achieve better compression, you can use OPTIMIZE TABLE or REPAIR TABLE. The number of rows in ARCHIVE tables reported by SHOW TABLE STATUS is always accurate. See [Section 13.5.2.5, “OPTIMIZE TABLE Syntax”](#), [Section 13.5.2.6, “REPAIR TABLE Syntax”](#), and [Section 13.5.4.21, “SHOW TABLE STATUS Syntax”](#).

Additional resources

- A forum dedicated to the ARCHIVE storage engine is available at <http://forums.mysql.com/list.php?112>.

14.9. The CSV Storage Engine

The CSV storage engine stores data in text files using comma-separated values format.

To enable this storage engine, use the `--with-csv-storage-engine` option to **configure** when you build MySQL.

The CSV storage engine is included in MySQL-Max binary distributions. To enable this storage engine if you build MySQL from source, invoke **configure** with the `--with-csv-storage-engine` option.

To examine the source for the CSV engine, look in the `sql/examples` directory of a MySQL source distribution.

When you create a CSV table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine also creates a data file. Its name begins with the table name and has a `.csv` extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = CSV;
```

```
Query OK, 0 rows affected (0.12 sec)
```

```
mysql> INSERT INTO test VALUES(1, 'record one'), (2, 'record two');
```

```
Query OK, 2 rows affected (0.00 sec)
```

```
Records: 2 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM test;
```

```
+-----+-----+
| i     | c           |
+-----+-----+
| 1     | record one  |
| 2     | record two  |
+-----+-----+
```

```
2 rows in set (0.00 sec)
```

If you examine the `test.csv` file in the database directory created by executing the preceding statements, its contents should look like this:

```
"1", "record one"  
"2", "record two"
```

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel or StarOffice Calc.

The CSV storage engine does not support indexing.

14.10. The BLACKHOLE Storage Engine

The BLACKHOLE storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;  
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> INSERT INTO test VALUES(1, 'record one'), (2, 'record two');  
Query OK, 2 rows affected (0.00 sec)  
Records: 2 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM test;  
Empty set (0.00 sec)
```

The BLACKHOLE storage engine is included in MySQL-Max binary distributions. To enable this storage engine if you build MySQL from source, invoke **configure** with the `--with-blackhole-storage-engine` option.

To examine the source for the BLACKHOLE engine, look in the `sql` directory of a MySQL source distribution.

When you create a BLACKHOLE table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. There are no other files associated with the table.

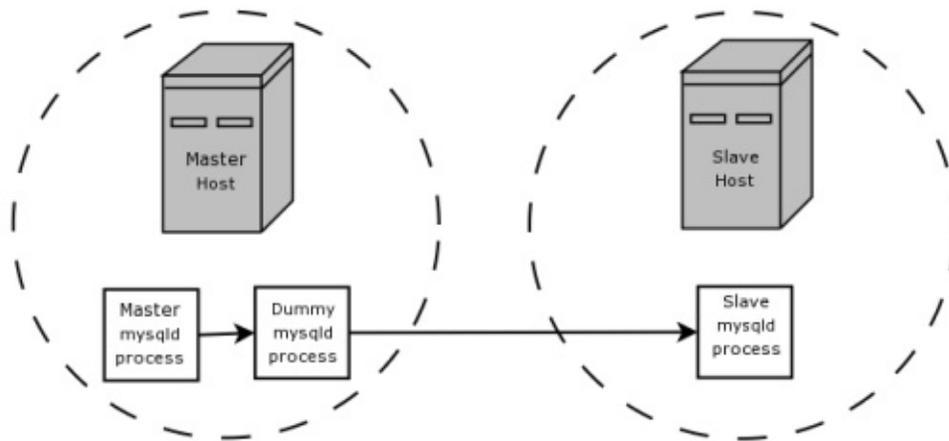
The BLACKHOLE storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

You can check whether the BLACKHOLE storage engine is available with this statement:

```
mysql> SHOW VARIABLES LIKE 'have_blackhole_engine';
```

Inserts into a BLACKHOLE table do not store any data, but if the binary log is enabled, the SQL statements are logged (and replicated to slave servers). This can be useful as a repeater or filter mechanism. For example, suppose that your application requires slave-side filtering rules, but transferring all binary log data to the slave first results in too much traffic. In such a case, it is possible to set up on the master host a “dummy” slave process whose default storage engine is

BLACKHOLE, depicted as follows:



The master writes to its binary log. The “dummy” **mysql** process acts as a slave, applying the desired combination of `replicate-do-*` and `replicate-ignore-*` rules, and writes a new, filtered binary log of its own. (See [Section 6.8, “Replication Startup Options”](#).) This filtered log is provided to the slave.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional **mysql** process on the replication master host. This type of setup can be repeated with additional replication slaves.

Other possible uses for the BLACKHOLE storage engine include:

- Verification of dump file syntax.
- Measurement of the overhead from binary logging, by comparing performance using BLACKHOLE with and without binary logging enabled.
- BLACKHOLE is essentially a “no-op” storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

Chapter 15. MySQL Cluster

Table of Contents

- [15.1. MySQL Cluster Overview](#)
- [15.2. Basic MySQL Cluster Concepts](#)
 - [15.2.1. MySQL Cluster Nodes, Node Groups, Replicas, and Partitions](#)
- [15.3. Simple Multi-Computer How-To](#)
 - [15.3.1. Hardware, Software, and Networking](#)
 - [15.3.2. Multi-Computer Installation](#)
 - [15.3.3. Multi-Computer Configuration](#)
 - [15.3.4. Initial Startup](#)
 - [15.3.5. Loading Sample Data and Performing Queries](#)
 - [15.3.6. Safe Shutdown and Restart](#)
- [15.4. MySQL Cluster Configuration](#)
 - [15.4.1. Building MySQL Cluster from Source Code](#)
 - [15.4.2. Installing the Software](#)
 - [15.4.3. Quick Test Setup of MySQL Cluster](#)
 - [15.4.4. Configuration File](#)
 - [15.4.5. Overview of Cluster Configuration Parameters](#)
 - [15.4.6. Configuring Parameters for Local Checkpoints](#)
- [15.5. Upgrading and Downgrading MySQL Cluster](#)
 - [15.5.1. Performing a Rolling Restart of the Cluster](#)
 - [15.5.2. Cluster Upgrade and Downgrade Compatibility](#)
- [15.6. Process Management in MySQL Cluster](#)
 - [15.6.1. MySQL Server Process Usage for MySQL Cluster](#)
 - [15.6.2. **ndbd**, the Storage Engine Node Process](#)
 - [15.6.3. **ndb mgmd**, the Management Server Process](#)
 - [15.6.4. **ndb_mgm**, the Management Client Process](#)
 - [15.6.5. Command Options for MySQL Cluster Processes](#)
- [15.7. Management of MySQL Cluster](#)
 - [15.7.1. MySQL Cluster Startup Phases](#)
 - [15.7.2. Commands in the Management Client](#)
 - [15.7.3. Event Reports Generated in MySQL Cluster](#)
 - [15.7.4. Single-User Mode](#)
- [15.8. On-line Backup of MySQL Cluster](#)
 - [15.8.1. Cluster Backup Concepts](#)

- [15.8.2. Using The Management Client to Create a Backup](#)
- [15.8.3. How to Restore a Cluster Backup](#)
- [15.8.4. Configuration for Cluster Backup](#)
- [15.8.5. Backup Troubleshooting](#)
- [15.9. Using High-Speed Interconnects with MySQL Cluster](#)
 - [15.9.1. Configuring MySQL Cluster to use SCI Sockets](#)
 - [15.9.2. Understanding the Impact of Cluster Interconnects](#)
- [15.10. Known Limitations of MySQL Cluster](#)
- [15.11. MySQL Cluster Development Roadmap](#)
 - [15.11.1. MySQL Cluster Changes in MySQL 5.0](#)
 - [15.11.2. MySQL 5.1 Development Roadmap for MySQL Cluster](#)
- [15.12. MySQL Cluster FAQ](#)
- [15.13. MySQL Cluster Glossary](#)

MySQL Cluster is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. It uses the NDB Cluster storage engine to enable running several MySQL servers in a cluster. This storage engine is available in MySQL 5.0 binary releases and in RPMs compatible with most modern Linux distributions. (If you install using RPM files, note that both the `mysql-server` and `mysql-max` RPMs must be installed to have MySQL Cluster capability.)

The operating systems on which MySQL Cluster is currently available are Linux, Mac OS X, and Solaris. (Some users have reported success with running MySQL Cluster on FreeBSD and HP-UX, although these platforms are not yet officially supported by MySQL AB.) We are working to make Cluster run on all operating systems supported by MySQL, including Windows, and will update this page as new platforms are supported.

This chapter represents a work in progress, and its contents are subject to revision as MySQL Cluster continues to evolve. Additional information regarding MySQL Cluster can be found on the MySQL AB Web site at <http://www.mysql.com/products/cluster/>.

Additional resources

- Answers to some commonly asked questions about Cluster may be found in the [Section 15.12, “MySQL Cluster FAQ”](#).

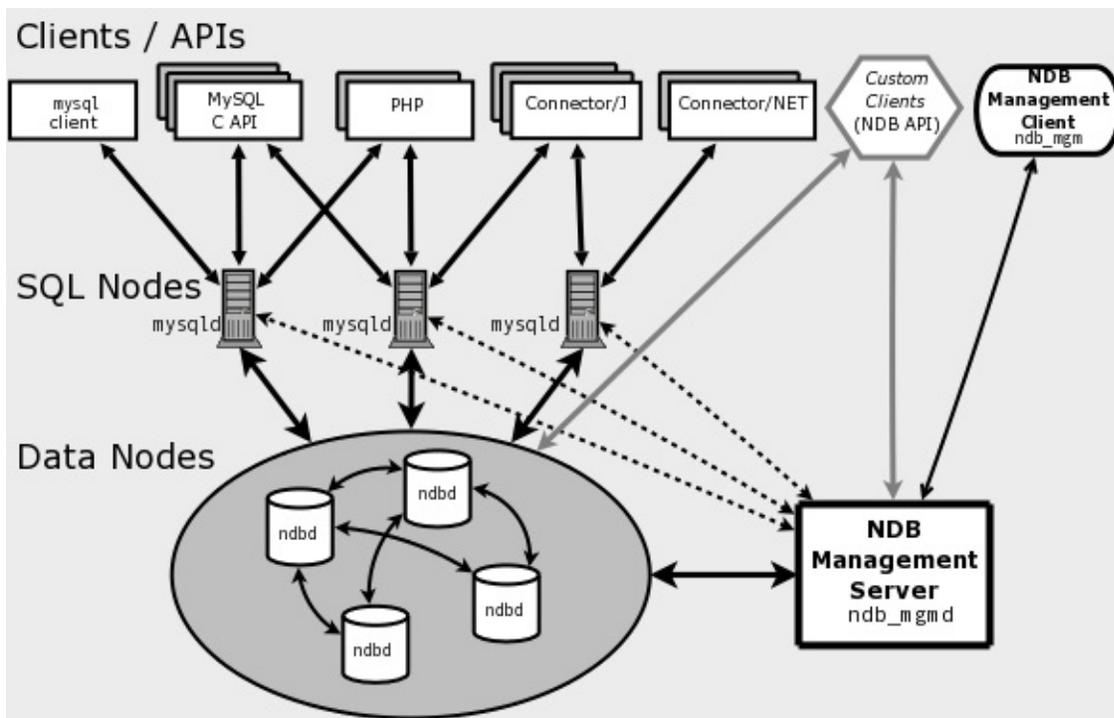
- The MySQL Cluster mailing list: <http://lists.mysql.com/cluster>.
- The MySQL Cluster Forum: <http://forums.mysql.com/list.php?25>.
- If you are new to MySQL Cluster, you may find our Developer Zone article [How to set up a MySQL Cluster for two servers](#) to be helpful.

15.1. MySQL Cluster Overview

MySQL Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture allows the system to work with very inexpensive hardware, and without any specific requirements on hardware or software. It also does not have any single point of failure because each component has its own memory and disk.

MySQL Cluster integrates the standard MySQL server with an in-memory clustered storage engine called NDB. In our documentation, the term NDB refers to the part of the setup that is specific to the storage engine, whereas “MySQL Cluster” refers to the combination of MySQL and the NDB storage engine.

A MySQL Cluster consists of a set of computers, each running a number of processes including MySQL servers, data nodes for NDB Cluster, management servers, and (possibly) specialized data access programs. The relationship of these components in a cluster is shown here:



All these programs work together to form a MySQL Cluster. When data is stored in the NDB cluster storage engine, the tables are stored in the data nodes. Such

tables are directly accessible from all other MySQL servers in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

The data stored in the data nodes for MySQL Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

15.2. Basic MySQL Cluster Concepts

NDB is an in-memory storage engine offering high-availability and data-persistence features.

The NDB storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. MySQL Cluster's NDB storage engine contains a complete set of data, dependent only on other data within the cluster itself.

The cluster portion of MySQL Cluster is currently configured independently of the MySQL servers. In a MySQL Cluster, each part of the cluster is considered to be a *node*.

Note: In many contexts, the term “node” is used to indicate a computer, but when discussing MySQL Cluster it means a *process*. There can be any number of nodes on a single computer, for which we use the term **cluster host**.

There are three types of cluster nodes, and in a minimal MySQL Cluster configuration, there will be at least three nodes, one of each of these types:

- The **management node** (MGM node): The role of this type of node is to manage the other nodes within the MySQL Cluster, such as providing configuration data, starting and stopping nodes, running backup, and so forth. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. An MGM node is started with the command **ndb_mgmd**.
- The **data node**: This is the type of node that stores the cluster's data. There are as many data nodes as there are replicas, times the number of fragments. For example, with two replicas, each having two fragments, you will need four data nodes. It is not necessary to have more than one replica. A data node is started with the command **ndbd**.
- The **SQL node**: This is the node that accesses the cluster data. In the case of MySQL Cluster, an SQL node is a traditional MySQL server that uses the NDB Cluster storage engine. An SQL node is typically started with the command **mysqld --ndbcluster** or by using **mysqld** with the `ndbcluster`

option added to `my.cnf`.

Important: It is not realistic to expect to employ a three-node setup in a production environment. Such a configuration provides no redundancy; in order to benefit from MySQL Cluster's high-availability features, you must use multiple data and SQL nodes. The use of multiple management nodes is also highly recommended.

For a brief introduction to the relationships between nodes, node groups, replicas, and partitions in MySQL Cluster, see [Section 15.2.1, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#).

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. MySQL Cluster is currently designed with the intention that data nodes are homogeneous in terms of processor power, memory space, and bandwidth. In addition, to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server (MGM node) manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the data nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These are of two types:

- **Standard MySQL clients:** These are no different for MySQL Cluster than they are for standard (non-Cluster) MySQL. In other words, MySQL Cluster can be accessed from existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on.
- **Management clients:** These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on.

15.2.1. MySQL Cluster Nodes, Node Groups, Replicas, and Partitions

This section discusses the manner in which MySQL Cluster divides and duplicates data for storage.

Central to an understanding of this topic are the following concepts, listed here with brief definitions:

- **(Data) Node:** An **ndbd** process, which stores a *replica* —that is, a copy of the *partition* (see below) assigned to the node group of which the node is a member.

Each data node should be located on a separate computer. While it is also possible to host multiple **ndbd** processes on a single computer, such a configuration is not supported.

It is common for the terms “node” and “data node” to be used interchangeably when referring to an **ndbd** process; where mentioned, management (MGM) nodes (**ndb_mgmd** processes) and SQL nodes (**mysqld** processes) are specified as such in this discussion.

- **Node Group:** A node group consists of one or more nodes, and stores partitions, or sets of *replicas* (see next item).

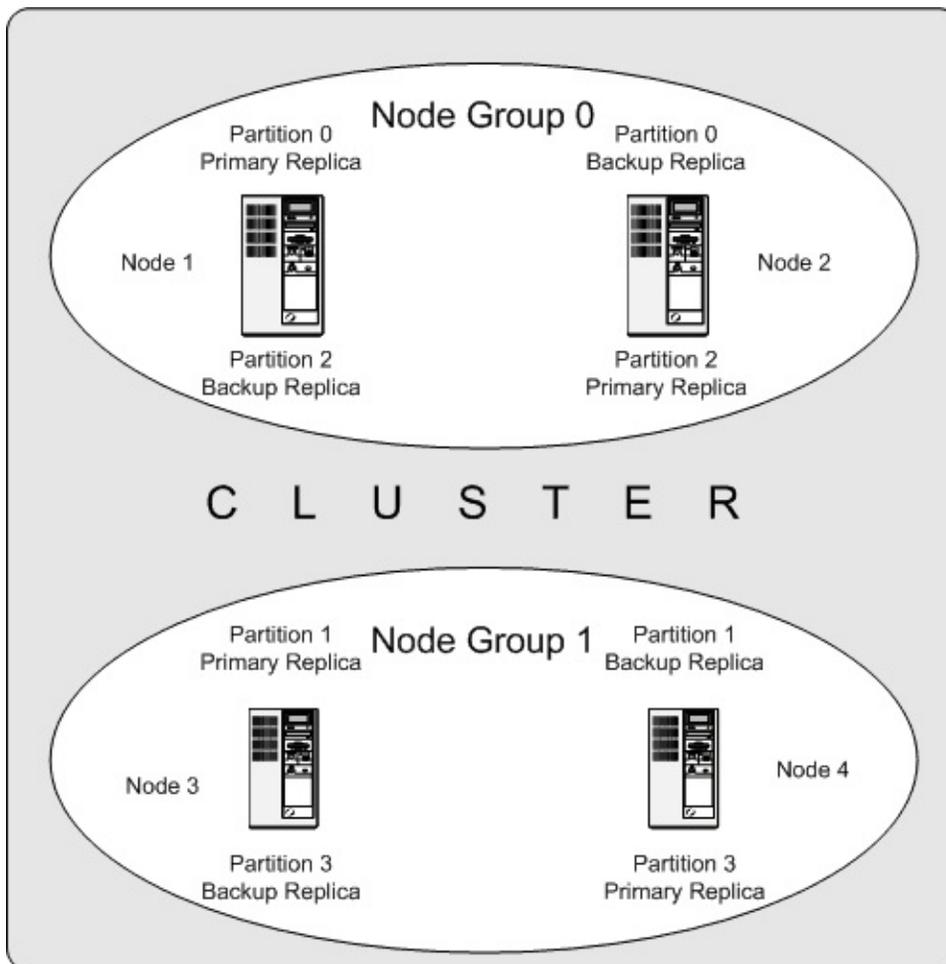
Note: Currently, all node groups in a cluster must have the same number of nodes.

- **Partition:** This is a portion of the data stored by the cluster. There are as many cluster partitions as nodes participating in the cluster. Each node is responsible for keeping at least one copy of any partitions assigned to it (that is, at least one replica) available to the cluster.

A replica belongs entirely to a single node; a node can (and usually does) store several replicas.

- **Replica:** This is a copy of a cluster partition. Each node in a node group stores a replica. Also sometimes known as a *partition replica*. The number of replicas is equal to the number of nodes per node group.

The following diagram illustrates a MySQL Cluster with four data nodes, arranged in two node groups of two nodes each; nodes 1 and 2 belong to node group 0, and nodes 3 and 4 belong to node group 1. Note that only data (**ndbd**) nodes are shown here; although a working cluster requires an **ndb_mgm** process for cluster management and at least one SQL node to access the data stored by the cluster, these have been omitted in the figure for clarity.



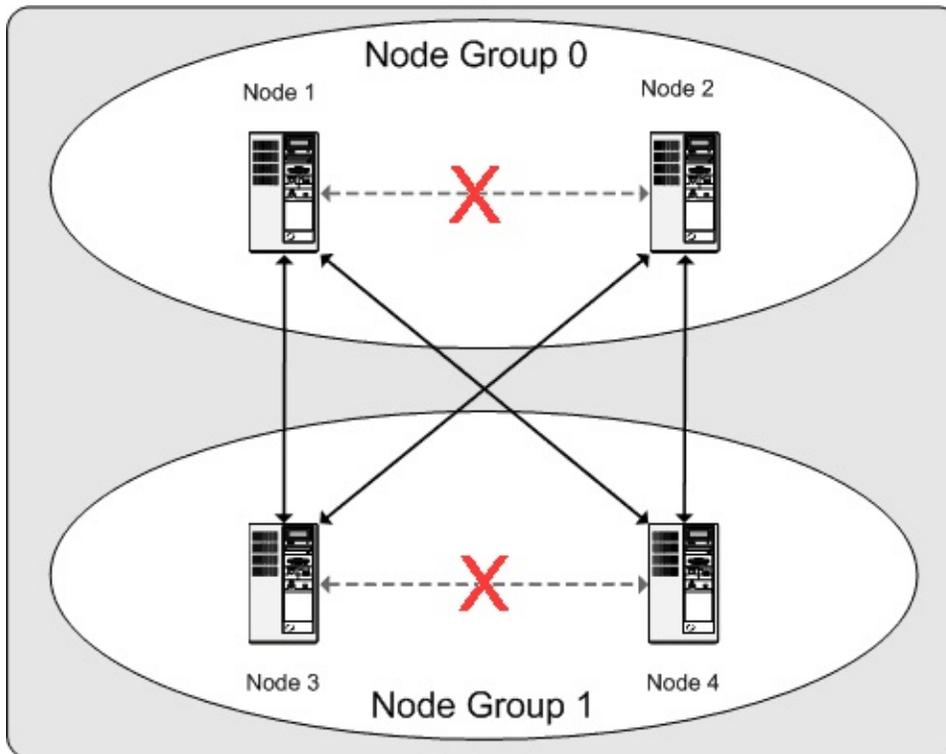
The data stored by the cluster is divided into four partitions, numbered 0, 1, 2, and 3. Each partition is stored — in multiple copies — on the same node group. Partitions are stored on alternate node groups: Partition 2 is stored on .

- Partition 0 is stored on node group 0; a *primary replica* (primary copy) is stored on node 1, and a *backup replica* (backup copy of the partition) is stored on node 2.
- Partition 1 is stored on the other node group (node group 1); this partition's

primary replica is on node 3, and its backup replica is on node 4.

- Partition 2 is stored on node group 0. However, the placing of its two replicas is reversed from that of Partition 0; for Partition 2, the primary replica is stored on node 2, and the backup on node 1.
- Partition 3 is stored on node group 1, and the placement of its two replicas are reversed from those of partition 1. That is, its primary replica is located on node 4, with the backup on node 3.

What this means regarding the continued operation of a MySQL Cluster is this: so long as each node group participating in the cluster has at least one node operating, the cluster has a complete copy of all data and remains viable. This is illustrated in the next diagram.



In this example, where the cluster consists of two node groups of two nodes each, any combination of at least one node in node group 0 and at least one node in node group 1 is sufficient to keep the cluster “alive” (indicated by arrows in the diagram). However, if *both* nodes from *either* node group fail, the remaining two nodes are not sufficient (shown by the arrows marked out with an **X**); in either case, the cluster has lost an entire partition and so can no longer provide

access to a complete set of all cluster data.

15.3. Simple Multi-Computer How-To

This section is a “How-To” that describes the basics for how to plan, install, configure, and run a MySQL Cluster. Whereas the examples in [Section 15.4, “MySQL Cluster Configuration”](#) provide more in-depth information on a variety of clustering options and configuration, the result of following the guidelines and procedures outlined here should be a usable MySQL Cluster which meets the *minimum* requirements for availability and safeguarding of data.

This section covers hardware and software requirements; networking issues; installation of MySQL Cluster; configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

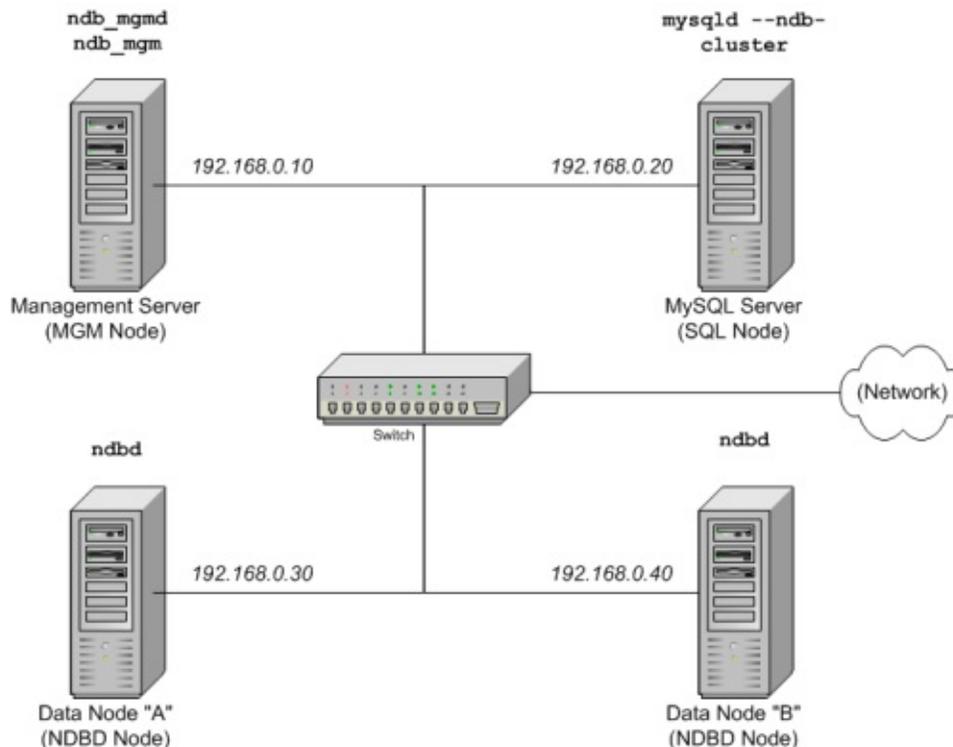
Basic Assumptions

This How-To makes the following assumptions:

1. The cluster setup has four nodes, each on a separate host, and each with a fixed network address on a typical Ethernet as shown here:

Node	IP Address
Management (MGM) node	192.168.0.10
MySQL server (SQL) node	192.168.0.20
Data (NDBD) node "A"	192.168.0.30
Data (NDBD) node "B"	192.168.0.40

This may be made clearer in the following diagram:



Note: In the interest of simplicity (and reliability), this How-To uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use hostnames in lieu of IP addresses in configuring Cluster. Alternatively, you can use the `/etc/hosts` file or your operating system's equivalent for providing a means to do host lookup if such is available.

2. Each host in our scenario is an Intel-based desktop PC running a common, generic Linux distribution installed to disk in a standard configuration, and running no unnecessary services. The core OS with standard TCP/IP networking capabilities should be sufficient. Also for the sake of simplicity, we also assume that the filesystems on all hosts are set up identically. In the event that they are not, you will need to adapt these instructions accordingly.
3. Standard 100 Mbps or 1 gigabit Ethernet cards are installed on each machine, along with the proper drivers for the cards, and that all four hosts are connected via a standard-issue Ethernet networking appliance such as a switch. (All machines should use network cards with the same throughput. That is, all four machines in the cluster should have 100 Mbps cards *or* all four machines should have 1 Gbps cards.) MySQL Cluster will work in a

100 Mbps network; however, gigabit Ethernet will provide better performance.

Note that MySQL Cluster is *not* intended for use in a network for which throughput is less than 100 Mbps. For this reason (among others), attempting to run a MySQL Cluster over a public network such as the Internet is not likely to be successful, and is not recommended.

4. For our sample data, we will use the `world` database which is available for download from the MySQL AB Web site. As this database takes up a relatively small amount of space, we assume that each machine has 256MB RAM, which should be sufficient for running the operating system, host NDB process, and (for the data nodes) for storing the database.

Although we refer to a Linux operating system in this How-To, the instructions and procedures that we provide here should be easily adaptable to either Solaris or Mac OS X. We also assume that you already know how to perform a minimal installation and configuration of the operating system with networking capability, or that you are able to obtain assistance in this elsewhere if needed.

We discuss MySQL Cluster hardware, software, and networking requirements in somewhat greater detail in the next section. (See [Section 15.3.1, “Hardware, Software, and Networking”](#).)

15.3.1. Hardware, Software, and Networking

One of the strengths of MySQL Cluster is that it can be run on commodity hardware and has no unusual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (Note that this is subject to change, and that we intend to implement disk-based storage in a future MySQL Cluster release.) Naturally, multiple and faster CPUs will enhance performance. Memory requirements for Cluster processes are relatively small.

The software requirements for Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support MySQL Cluster. For Mac OS X or Solaris, the standard installation is sufficient. For Linux, a standard, “out of the box” installation should be all that is necessary. The MySQL software requirements are simple: all that is needed is

a production release of MySQL-max 5.0; you must use the -max version of MySQL to have Cluster support. (See [Section 5.3, “The `mysqld-max` Extended MySQL Server”](#).) It is not necessary to compile MySQL yourself merely to be able to use Cluster. In this How-To, we assume that you are using the -max binary appropriate to your Linux, Solaris, or Mac OS X operating system, available via the MySQL software downloads page at <http://dev.mysql.com/downloads/>.

For inter-node communication, Cluster supports TCP/IP networking in any standard topology, and the minimum expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or router to provide network connectivity for the cluster as a whole. We strongly recommend that a MySQL Cluster be run on its own subnet which is not shared with non-Cluster machines for the following reasons:

- **Security:** Communications between Cluster nodes are not encrypted or shielded in any way. The only means of protecting transmissions within a MySQL Cluster is to run your Cluster on a protected network. If you intend to use MySQL Cluster for Web applications, the cluster should definitely reside behind your firewall and not in your network's De-Militarized Zone ([DMZ](#)) or elsewhere.
- **Efficiency:** Setting up a MySQL Cluster on a private or protected network allows the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch for your MySQL Cluster not only helps protect against unauthorized access to Cluster data, it also ensures that Cluster nodes are shielded from interference caused by transmissions between other computers on the network. For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

It is also possible to use the high-speed Scalable Coherent Interface (SCI) with MySQL Cluster, but this is not a requirement. See [Section 15.9, “Using High-Speed Interconnects with MySQL Cluster”](#), for more about this protocol and its use with MySQL Cluster.

15.3.2. Multi-Computer Installation

Each MySQL Cluster host computer running data or SQL nodes must have

installed on it a MySQL-max binary. For management nodes, it is not necessary to install the MySQL server binary, but you do have to install the MGM server daemon and client binaries (**ndb_mgmd** and **ndb_mgm**, respectively). This section covers the steps necessary to install the correct binaries for each type of Cluster node.

MySQL AB provides precompiled binaries that support Cluster, and there is generally no need to compile these yourself. Therefore, the first step in the installation process for each cluster host is to download the file `mysql-max-5.0.25-pc-linux-gnu-i686.tar.gz` from the [MySQL downloads area](#). We assume that you have placed it in each machine's `/var/tmp` directory. (If you do require a custom binary, see [Section 2.9.3, “Installing from the Development Source Tree”](#).)

RPMs are also available for both 32-bit and 64-bit Linux platforms; as of MySQL 4.1.10a, the -max binaries installed by the RPMs support the NDBCluster storage engine. If you choose to use these rather than the binary files, be aware that you must install *both* the -server and -max packages on all machines that are to host cluster nodes. (See [Section 2.4, “Installing MySQL on Linux”](#), for more information about installing MySQL using the RPMs.) After installing from RPM, you will still need to configure the cluster as discussed in [Section 15.3.3, “Multi-Computer Configuration”](#).

Note: After completing the installation, do not yet start any of the binaries. We will show you how to do so following the configuration of all nodes.

Storage and SQL Node Installation

On each of the three machines designated to host storage or SQL nodes, perform the following steps as the system root user:

1. Check your `/etc/passwd` and `/etc/group` files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a `mysql` group and `mysql` user on the system. Some OS distributions create these as part of the operating system installation process. If they are not already present, create a new `mysql` user group, and then add a `mysql` user to this group:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
```

The syntax for **useradd** and **groupadd** may differ slightly on different versions of Unix, or they may have different names such as **adduser** and **addgroup**.

2. Change location to the directory containing the downloaded file, unpack the archive, and create a symlink to the `mysql-max` directory named `mysql`. Note that the actual file and directory names will vary according to the MySQL version number.

```
shell> cd /var/tmp
shell> tar -xzvf -C /usr/local mysql-max-5.0.25-pc-linux-gnu-i686
shell> ln -s /usr/local/mysql-max-5.0.25-pc-linux-gnu-i686 /usr/
```

3. Change location to the `mysql` directory and run the supplied script for creating the system databases:

```
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
```

4. Set the necessary permissions for the MySQL server and data directories:

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

Note that the data directory on each machine hosting a data node is `/usr/local/mysql/data`. We will use this piece of information when we configure the management node. (See [Section 15.3.3, “Multi-Computer Configuration”](#).)

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

```
shell> cp support-files/mysql.server /etc/rc.d/init.d/
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server
```

(The startup scripts directory may vary depending on your operating system and version — for example, in some Linux distributions, it is `/etc/init.d`.)

Here we use Red Hat's **chkconfig** for creating links to the startup scripts;

use whatever means is appropriate for this purpose on your operating system and distribution, such as **update-rc.d** on Debian.

Remember that the preceding steps must be performed separately for each machine on which a storage or SQL node is to reside.

Management Node Installation

Installation for the management (MGM) node does not require installation of the **mysqld** binary. Only the binaries for the MGM server and client are required, which can be found in the downloaded archive. Again, we assume that you have placed this file in `/var/tmp`.

As system root (that is, after using **sudo**, **su root**, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install **ndb_mgmd** and **ndb_mgm** on the Cluster management node host:

1. Change location to the `/var/tmp` directory, and extract the **ndb_mgm** and **ndb_mgmd** from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-5.0.25-pc-linux-gnu-i686.tar.gz
shell> cd mysql-5.0.25-pc-linux-gnu-i686
shell> cp /bin/ndb_mgm* /usr/local/bin
```

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once **ndb_mgm** and **ndb_mgmd** have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

In [Section 15.3.3, “Multi-Computer Configuration”](#), we will create and write configuration files for all of the nodes in our example Cluster.

15.3.3. Multi-Computer Configuration

For our four-node, four-host MySQL Cluster, we will need to write four configuration files, one per node/host.

- Each data node or SQL node requires a `my.cnf` file that provides two pieces of information: a **connectstring** telling the node where to find the MGM node, and a line telling the MySQL server on this host (the machine hosting the data node) to run in NDB mode.

For more information on connectstrings, see [Section 15.4.4.2, “The Cluster connectstring”](#).

- The management node needs a `config.ini` file telling it how many replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the Storage and SQL Nodes

The `my.cnf` file needed for the data nodes is fairly simple. The configuration file should be located in the `/etc` directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

```
shell> vi /etc/my.cnf
```

We show `vi` being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, `my.cnf` should look like this:

```
# Options for mysqld process:
[MYSQLD]
ndbcluster                # run NDB engine
ndb-connectstring=192.168.0.10 # location of MGM node

# Options for ndbd process:
[MYSQL_CLUSTER]
ndb-connectstring=192.168.0.10 # location of MGM node
```

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.

Important: Once you have started a **mysqld** process with the `ndbcluster` and `ndb-connectstring` parameters in the `[MYSQLD]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design.*

Configuring the Management Node

The first step in configuring the MGM node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as root):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our representative setup, the `config.ini` file should read as follows:

```
# Options affecting ndbd processes on all data nodes:
[NDBD DEFAULT]
NoOfReplicas=2      # Number of replicas
DataMemory=80M     # How much memory to allocate for data storage
IndexMemory=18M    # How much memory to allocate for index storage
                   # For DataMemory and IndexMemory, we have used the
                   # default values. Since the "world" database takes
                   # only about 500KB, this should be more than enoug
                   # this example Cluster setup.

# TCP/IP options:
[TCP DEFAULT]
portnumber=2202    # This the default; however, you can use any
                   # port that is free for all the hosts in cluster
                   # Note: It is recommended beginning with MySQL 5.0
                   # you do not specify the portnumber at all and sim
                   # the default value to be used instead

# Management process options:
[NDB_MGMD]
hostname=192.168.0.10 # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster # Directory for MGM node logfiles

# Options for data node "A":
[NDBD]
                               # (one [NDBD] section per data node)
hostname=192.168.0.30 # Hostname or IP address
datadir=/usr/local/mysql/data # Directory for this data node's dat
```

```

# Options for data node "B":
[NDBD]
hostname=192.168.0.40      # Hostname or IP address
datadir=/usr/local/mysql/data # Directory for this data node's dat

# SQL node options:
[MYSQLD]
hostname=192.168.0.20      # Hostname or IP address
                             # (additional mysqld connections can
                             # specified for this node for variou
                             # purposes such as running ndb_resto

```

(**Note:** The world database can be downloaded from <http://dev.mysql.com/doc/>, where it can be found listed under “Examples.”)

After all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss how this is done in [Section 15.3.4, “Initial Startup”](#).

For more detailed information about the available MySQL Cluster configuration parameters and their uses, see [Section 15.4.4, “Configuration File”](#), and [Section 15.4, “MySQL Cluster Configuration”](#). For configuration of MySQL Cluster as relates to making backups, see [Section 15.8.4, “Configuration for Cluster Backup”](#).

Note: The default port for Cluster management nodes is 1186; the default port for data nodes is 2202. Beginning with MySQL 5.0.3, this restriction is lifted, and the cluster automatically allocates ports for data nodes from those that are already free.

15.3.4. Initial Startup

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. Although it is possible to start the nodes in any order, it is recommended that the management node be started first, followed by the storage nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the MGM node process:

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

Note that **ndb_mgmd** must be told where to find its configuration file, using the `-f` or `--config-file` option. (See [Section 15.6.3, “ndb_mgmd, the Management Server Process”](#), for details.)

2. On each of the data node hosts, run this command to start the **ndbd** process for the first time:

```
shell> ndbd --initial
```

Note that it is very important to use the `--initial` parameter *only* when starting **ndbd** for the first time, or when restarting after a backup/restore operation or a configuration change. This is because the `--initial` option causes the node to delete any files created by earlier **ndbd** instances that are needed for recovery, including the recovery log files.

3. If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the startup script installed in `/etc/init.d` to start the MySQL server process on the SQL node. Note that you need to install the `-max` server RPM *in addition to* the Standard server RPM to run the `-max` server binary.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the **ndb_mgm** management node client. The output should look like that shown here, although you might see some slight differences in the output depending upon the exact version of MySQL that you are using:

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)]      2 node(s)
id=2      @192.168.0.30 (Version: 5.0.25, Nodegroup: 0, Master)
id=3      @192.168.0.40 (Version: 5.0.25, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1      @192.168.0.10 (Version: 5.0.25)

[mysqld(SQL)]   1 node(s)
```

id=4 (Version: 5.0.25)

Note: If you are using an older version of MySQL, you may see the SQL node referenced as `[mysqld(API)]`. This reflects an older usage that is now deprecated.

You should now be ready to work with databases, tables, and data in MySQL Cluster. See [Section 15.3.5, “Loading Sample Data and Performing Queries”](#), for a brief discussion.

15.3.5. Loading Sample Data and Performing Queries

Working with data in MySQL Cluster is not much different from doing so in MySQL without Cluster. There are two points to keep in mind:

- For a table to be replicated in the cluster, it must use the `NDB Cluster` storage engine. To specify this, use the `ENGINE=NDB` or `ENGINE=NDBCLUSTER` table option. You can add this option when creating the table:

```
CREATE TABLE tbl_name ( ... ) ENGINE=NDBCLUSTER;
```

Alternatively, for an existing table that uses a different storage engine, use `ALTER TABLE` to change the table to use `NDB Cluster`:

```
ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
```

- Each `NDB` table *must* have a primary key. If no primary key is defined by the user when a table is created, the `NDB Cluster` storage engine automatically generates a hidden one. (**Note:** This hidden key takes up space just as does any other table index. It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of **mysqldump**, you can open the SQL script in a text editor and add the `ENGINE` option to any table creation statements, or replace any existing `ENGINE` (or `TYPE`) options. Suppose that you have the `world` sample database on another MySQL server that does not support MySQL Cluster, and you want to export the `City` table:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

The resulting `city_table.sql` file will contain this table creation statement (and the `INSERT` statements necessary to import the table data):

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabul',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

You will need to make sure that MySQL uses the NDB storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition *before* importing it into the Cluster database. Using the `city` table as an example, modify the `ENGINE` option of the definition as follows:

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabul',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of `TYPE=engine_name` or `ENGINE=engine_name` with `ENGINE=NDBCLUSTER`. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use `ALTER TABLE` to change their storage engine. The particulars are

given later in this section.

Assuming that you have already created a database named `world` on the SQL node of the cluster, you can then use the **mysql** command-line client to read `city_table.sql`, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address `192.168.0.20`).

To create a copy of the entire `world` database on the SQL node, use **mysqldump** on the non-cluster server to export the database to a file named `world.sql`; for example, in the `/tmp` directory. Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

```
shell> mysql world < /tmp/world.sql
```

If you save the file to a different location, adjust the preceding instructions accordingly.

It is important to note that `NDB Cluster` in MySQL 5.0 does not support autodiscovery of databases. (See [Section 15.10, “Known Limitations of MySQL Cluster”](#).) This means that, once the `world` database and its tables have been created on one data node, you need to issue the `CREATE SCHEMA world` statement (beginning with MySQL 5.0.2, you may use `CREATE SCHEMA world` instead), followed by `FLUSH TABLES` on each SQL node in the cluster. This causes the node to recognize the database and read its table definitions.

Running `SELECT` queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the root password at the `Enter password:` prompt):

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.25
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql>
```

We simply use the MySQL server's root account and assume that you have followed the standard security precautions for installing a MySQL server, including setting a strong root password. For more information, see [Section 2.10.3, “Securing the Initial MySQL Accounts”](#).

It is worth taking into account that Cluster nodes do not make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the root account) effects only applications that access the SQL node, not interaction between nodes.

If you did not modify the `ENGINE` clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Selecting a database and running a `SELECT` query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LI
+-----+-----+
| Name      | Population |
+-----+-----+
| Bombay    | 10500000  |
| Seoul     | 9981619   |
| São Paulo | 9968485   |
| Shanghai  | 9696300   |
| Jakarta   | 9604900   |
+-----+-----+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

Applications that use MySQL can employ standard APIs to access NDB tables. It is important to remember that your application must access the SQL node, and

not the MGM or data nodes. This brief example shows how we might execute the SELECT statement just shown by using PHP 5's mysqli extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('192.168.0.20', 'root', 'root_password', 'world')
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";

  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing="1">
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
<?
  # then display the results...
  while($row = $result->fetch_object())
    printf(<tr>\n  <td align="center">%s</td><td>%d</td>\n</tr>\
          $row->Name, $row->Population);
?>
  </tbody>
</table>
<?
  # ...and verify the number of rows that were retrieved
  printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
  }
```

```

else
  # otherwise, tell us what went wrong
  echo mysqli_error();

# free the result set and the mysqli connection object
$result->close();
$link->close();
?>
</body>
</html>

```

We assume that the process running on the Web server can reach the IP address of the SQL node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL AB's own Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

15.3.6. Safe Shutdown and Restart

To shut down the cluster, enter the following command in a shell on the machine hosting the MGM node:

```
shell> ndb_mgm -e shutdown
```

The `-e` option here is used to pass a command to the **ndb_mgm** client from the shell. See [Section 4.3.1, “Using Options on the Command Line”](#). The command causes the **ndb_mgm**, **ndb_mgmd**, and any **ndbd** processes to terminate gracefully. Any SQL nodes can be terminated using **mysqladmin shutdown** and other means.

To restart the cluster, run these commands:

- On the management host (192.168.0.10 in our example setup):

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

- On each of the data node hosts (192.168.0.30 and 192.168.0.40):

```
shell> ndbd
```

Remember *not* to invoke this command with the `--initial` option when restarting an NDBD node normally.

- On the SQL host (192.168.0.20):

```
shell> mysqld &
```

For information on making Cluster backups, see [Section 15.8.2, “Using The Management Client to Create a Backup”](#).

To restore the cluster from backup requires the use of the **ndb_restore** command. This is covered in [Section 15.8.3, “How to Restore a Cluster Backup”](#).

More information on configuring MySQL Cluster can be found in [Section 15.4, “MySQL Cluster Configuration”](#).

15.4. MySQL Cluster Configuration

A MySQL server that is part of a MySQL Cluster differs in only one respect from a normal (non-clustered) MySQL server, in that it employs the NDB Cluster storage engine. This engine is also referred to simply as NDB, and the two forms of the name are synonymous.

To avoid unnecessary allocation of resources, the server is configured by default with the NDB storage engine disabled. To enable NDB, you must modify the server's `my.cnf` configuration file, or start the server with the `--ndbcluster` option.

The MySQL server is a part of the cluster, so it also must know how to access an MGM node to obtain the cluster configuration data. The default behavior is to look for the MGM node on `localhost`. However, should you need to specify that its location is elsewhere, this can be done in `my.cnf` or on the MySQL server command line. Before the NDB storage engine can be used, at least one MGM node must be operational, as well as any desired data nodes.

15.4.1. Building MySQL Cluster from Source Code

NDB, the Cluster storage engine, is available in binary distributions for Linux, Mac OS X, and Solaris. We are working to make Cluster run on all operating systems supported by MySQL, including Windows.

If you choose to build from a source tarball or the MySQL 5.0 BitKeeper tree, be sure to use the `--with-ndbcluster` option when running **configure**. You can also use the **BUILD/compile-pentium-max** build script. Note that this script includes OpenSSL, so you must either have or obtain OpenSSL to build successfully, or else modify **compile-pentium-max** to exclude this requirement. Of course, you can also just follow the standard instructions for compiling your own binaries, and then perform the usual tests and installation procedure. See [Section 2.9.3, “Installing from the Development Source Tree”](#).

15.4.2. Installing the Software

In the next few sections, we assume that you are already familiar with installing

MySQL, and here we cover only the differences between configuring MySQL Cluster and configuring MySQL without clustering. (See [Chapter 2, *Installing and Upgrading MySQL*](#), if you require more information about the latter.)

You will find Cluster configuration easiest if you have already have all management and data nodes running first; this is likely to be the most time-consuming part of the configuration. Editing the `my.cnf` file is fairly straightforward, and this section will cover only any differences from configuring MySQL without clustering.

15.4.3. Quick Test Setup of MySQL Cluster

To familiarize you with the basics, we will describe the simplest possible configuration for a functional MySQL Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as `/var/lib/mysql-cluster`, by executing the following command as the system root user:

```
shell> mkdir /var/lib/mysql-cluster
```

In this directory, create a file named `config.ini` that contains the following information. Substitute appropriate values for `HostName` and `DataDir` as necessary for your system.

```
# file "config.ini" - showing minimal setup consisting of 1 data nod
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only fo
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not requi
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommen
# set it explicitly.
# Note: DB, API, and MGM are aliases for NDBD, MYSQLD, and NDB_MGMD
# respectively. DB and API are deprecated and should not be used in
# installations.
[NDBD DEFAULT]
NoOfReplicas= 1

[MYSQLD DEFAULT]
[NDB_MGMD DEFAULT]
```

```
[TCP DEFAULT]
```

```
[NDB_MGMD]
```

```
HostName= myhost.example.com
```

```
[NDBD]
```

```
HostName= myhost.example.com
```

```
DataDir= /var/lib/mysql-cluster
```

```
[MYSQLD]
```

```
[MYSQLD]
```

```
[MYSQLD]
```

You can now start the **ndb_mgmd** management server. By default, it attempts to read the `config.ini` file in its current working directory, so change location into the directory where the file is located and then invoke **ndb_mgmd**:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Then start a single data node by running **ndbd**. When starting **ndbd** for a given data node for the very first time, you should use the `--initial` option as shown here:

```
shell> ndbd --initial
```

For subsequent **ndbd** starts, you will generally want to *omit* the `--initial` option:

```
shell> ndbd
```

The reason for omitting `--initial` on subsequent restarts is that this option causes **ndbd** to delete and re-create all existing data and log files (as well as all table metadata) for this data node. One exception to this rule about not using `--initial` except for the first **ndbd** invocation is that you use it when restarting the cluster and restoring from backup after adding new data nodes.

By default, **ndbd** looks for the management server at `localhost` on port 1186.

Note: If you have installed MySQL from a binary tarball, you will need to specify the path of the **ndb_mgmd** and **ndbd** servers explicitly. (Normally, these will be found in `/usr/local/mysql/bin.`)

Finally, change location to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```
shell> mysqld_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

If all has gone well so far, you now can start using the cluster. Connect to the server and verify that the NDBCLUSTER storage engine is enabled:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.25-Max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
***** 12. row *****
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
***** 13. row *****
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...

```

The row numbers shown in the preceding example output may be different from those shown on your system, depending upon how your server is configured.

Try to create an NDBCLUSTER table:

```
shell> mysql
mysql> USE test;
Database changed
```

```
mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> SHOW CREATE TABLE ctest \G
***** 1. row *****
      Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

To check that your nodes were set up properly, start the management client:

```
shell> ndb_mgm
```

Use the **SHOW** command from within the management client to obtain a report on the cluster's status:

```
NDB> SHOW
Cluster Configuration
-----
[ndbd(NDB)]      1 node(s)
id=2      @127.0.0.1 (Version: 3.5.3, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1      @127.0.0.1 (Version: 3.5.3)

[mysqld(API)]   3 node(s)
id=3      @127.0.0.1 (Version: 3.5.3)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

At this point, you have successfully set up a working MySQL Cluster. You can now store data in the cluster by using any table created with ENGINE=NDBCLUSTER or its alias ENGINE=NDB.

15.4.4. Configuration File

Configuring MySQL Cluster requires working with two files:

- `my.cnf`: Specifies options for all MySQL Cluster executables. This file, with which you should be familiar with from previous work with MySQL, must be accessible by each executable running in the cluster.
- `config.ini`: This file is read only by the MySQL Cluster management

server, which then distributes the information contained therein to all processes participating in the cluster. `config.ini` contains a description of each node involved in the cluster. This includes configuration parameters for data nodes and configuration parameters for connections between all nodes in the cluster. For a quick reference to the sections that can appear in this file, and what sorts of configuration parameters may be placed in each section, see [Sections of the config.ini File](#).

We are continuously making improvements in Cluster configuration and attempting to simplify this process. Although we strive to maintain backward compatibility, there may be times when introduce an incompatible change. In such cases we will try to let Cluster users know in advance if a change is not backward compatible. If you find such a change and we have not documented it, please report it in the MySQL bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#).

15.4.4.1. Basic Example Configuration

To support MySQL Cluster, you will need to update `my.cnf` as shown in the following example. Note that the options shown here should not be confused with those that are used in `config.ini` files. You may also specify these parameters on the command line when invoking the executables.

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid in MySQL 5.0)

# enable ndbcluster storage engine, and provide connectstring for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 11
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 11
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
```

```
[ndb_mgmd]
config-file=/etc/config.ini
```

(For more information on connectstrings, see [Section 15.4.4.2, “The Cluster connectstring”](#).)

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connectstring for ma
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Important: Once you have started a **mysqld** process with the `ndbcluster` and `ndb-connectstring` parameters in the `[MYSQLD]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design.*

You may also use a separate `[mysql_cluster]` section in the cluster `my.cnf` file for settings to be read and used by all executables:

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

For additional NDB variables that can be set in the `my.cnf` file, see [Section 5.2.2, “Server System Variables”](#).

The configuration file is named `config.ini` by default. It is read by **ndb_mgmd** at startup and can be placed anywhere. Its location and name are specified by using `--config-file=path_name` on the **ndb_mgmd** command line. If the configuration file is not specified, **ndb_mgmd** by default tries to read a file named `config.ini` located in the current working directory.

Currently, the configuration file is in INI format, which consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard INI format is that the parameter name and value can be separated by a colon (‘:’) as well as the equals sign (‘=’). Another deviation is that sections are not uniquely

identified by section name. Instead, unique sections (such as two different nodes of the same type) are identified by a unique ID specified as a parameter within the section.

Default values are defined for most parameters, and can also be specified in `config.ini`. To create a default value section, simply add the word `DEFAULT` to the section name. For example, an `[NDBD]` section contains parameters that apply to a particular data node, whereas an `[NDBD DEFAULT]` section contains parameters that apply to all data nodes. Suppose that all data nodes should use the same data memory size. To configure them all, create an `[NDBD DEFAULT]` section that contains a `DataMemory` line to specify the data memory size.

At a minimum, the configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two data nodes and two MySQL servers is shown here:

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com
```

```
[NDBD DEFAULT]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster
```

```
[NDB_MGMD]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster
```

```
[NDBD]
HostName= ndbd_2.mysql.com
```

```
[NDBD]
HostName= ndbd_3.mysql.com
```

```
[MYSQLD]
[MYSQLD]
HostName= mysqld_5.mysql.com
```

Note that each node has its own section in the `config.ini`. For instance, this cluster has two data nodes, so the preceding configuration file contains two `[NDBD]` sections defining these nodes.

Sections of the config.ini File

There are six different sections that you can use in the config.ini configuration file, as described in the following list:

- [COMPUTER]: Defines cluster hosts. This is not required to configure a viable MySQL Cluster, but it may be used as a convenience when setting up a large cluster. See [Section 15.4.4.3, “Defining Cluster Computers”](#), for more information.
- [NDBD]: Defines a cluster data node (**ndbd** process). See [Section 15.4.4.5, “Defining Data Nodes”](#), for details.
- [MYSQLD]: Defines the cluster's MySQL server nodes (also called SQL or API nodes). For a discussion of SQL node configuration, see [Section 15.4.4.6, “Defining SQL Nodes”](#).
- [MGM] or [NDB_MGMD]: Defines a cluster management server (MGM) node. For information concerning the configuration of MGM nodes, see [Section 15.4.4.4, “Defining the Management Server”](#).
- [TCP]: Defines a TCP/IP connection between cluster nodes, with TCP/IP being the default connection protocol. Normally, [TCP] or [TCP DEFAULT] sections are not required to set up a MySQL Cluster, as the cluster handles this automatically; however, it may be necessary in some situations to override the defaults provided by the cluster. See [Section 15.4.4.7, “Cluster TCP/IP Connections”](#), for information about available TCP/IP configuration parameters and how to use them. (You may also find [Section 15.4.4.8, “TCP/IP Connections Using Direct Connections”](#) to be of interest in some cases.)
- [SHM]: Defines shared-memory connections between nodes. In MySQL 5.0-max, it is enabled by default, but should still be considered experimental. For a discussion of SHM interconnects, see [Section 15.4.4.9, “Shared-Memory Connections”](#).
- [SCI]: Defines *Scalable Coherent Interface* connections between cluster data nodes. Such connections require software which, while freely available, is not part of the MySQL Cluster distribution, as well as specialised hardware. See [Section 15.4.4.10, “SCI Transport Connections”](#)

for detailed information about SCI interconnects.

You can define `DEFAULT` values for each section. All Cluster parameter names are case-insensitive, which differs from parameters specified in `my.cnf` or `my.ini` files.

15.4.4.2. The Cluster connectstring

With the exception of the MySQL Cluster management server (**`ndb_mgmd`**), each node that is part of a MySQL Cluster requires a *connectstring* that points to the management server's location. This connectstring is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connectstring is as follows:

```
<connectstring> :=  
    [<nodeid-specification>, ]<host-specification>[, <host-specificati
```

```
<nodeid-specification> := node_id
```

```
<host-specification> := host_name[:port_num]
```

node_id is an integer larger than 1 which identifies a node in `config.ini`.

host_name is a string representing a valid Internet host name or IP address.

port_num is an integer referring to a TCP/IP port number.

```
example 1 (long):      "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3
```

```
example 2 (short):    "myhost1"
```

All nodes will use `localhost:1186` as the default connectstring value if none is provided. If *port_num* is omitted from the connectstring, the default port is 1186. This port should always be available on the network because it has been assigned by IANA for this purpose (see <http://www.iana.org/assignments/port-numbers> for details).

By listing multiple `<host-specification>` values, it is possible to designate several redundant management servers. A cluster node will attempt to contact successive management servers on each host in the order specified, until a successful connection has been established.

There are a number of different ways to specify the connectstring:

- Each executable has its own command-line option which enables specifying the management server at startup. (See the documentation for the respective executable.)
- It is also possible to set the connectstring for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server's `my.cnf` file.
- For backward compatibility, two other options are available, using the same syntax:
 1. Set the `NDB_CONNECTSTRING` environment variable to contain the connectstring.
 2. Write the connectstring for each executable into a text file named `Ndb.cfg` and place this file in the executable's startup directory.

However, these are now deprecated and should not be used for new installations.

The recommended method for specifying the connectstring is to set it on the command line or in the `my.cnf` file for each executable.

15.4.4.3. Defining Cluster Computers

The `[COMPUTER]` section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- `Id`

This is an integer value, used to refer to the host computer elsewhere in the configuration file. This is not the same as the node ID.

- `HostName`

This is the computer's hostname or IP address.

15.4.4.4. Defining the Management Server

The [NDB_MGMD] section is used to configure the behavior of the management server. [MGM] can be used as an alias; the two section names are equivalent. All parameters in the following list are optional and assume their default values if omitted. **Note:** If neither the `ExecuteOnComputer` nor the `HostName` parameter is present, the default value `localhost` will be assumed for both.

- `Id`

Each node in the cluster has a unique identity, which is represented by an integer value in the range 1 to 63 inclusive. This ID is used by all internal cluster messages for addressing the node.

- `ExecuteOnComputer`

This refers to the `Id` set for one of the computers defined in a [COMPUTER] section of the `config.ini` file.

- `PortNumber`

This is the port number on which the management server listens for configuration requests and management commands.

- `HostName`

Specifying this parameter defines the hostname of the computer on which the management node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `LogDestination`

This parameter specifies where to send cluster logging information. There are three options in this regard: `CONSOLE`, `SYSLOG`, and `FILE`:

- `CONSOLE` outputs the log to `stdout`:

`CONSOLE`

- `SYSLOG` sends the log to a `syslog` facility, possible values being one of `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, or `local7`.

Note: Not every facility is necessarily supported by every operating system.

```
SYSLOG:facility=syslog
```

- FILE pipes the cluster log output to a regular file on the same machine. The following values can be specified:
 - filename: The name of the logfile.
 - maxsize: The maximum size (in bytes) to which the file can grow before logging rolls over to a new file. When this occurs, the old logfile is renamed by appending *.N* to the filename, where *N* is the next number not yet used with this name.
 - maxfiles: The maximum number of logfiles.

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

It is possible to specify multiple log destinations separated by semicolons as shown here:

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

The default value for the FILE parameter is

```
FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=
```

where *node_id* is the ID of the node.

- ArbitrationRank

This parameter is used to define which nodes can act as arbitrators. Only MGM nodes and SQL nodes can be arbitrators. ArbitrationRank can take one of the following values:

- 0: The node will never be used as an arbitrator.
- 1: The node has high priority; that is, it will be preferred as an arbitrator over low-priority nodes.
- 2: Indicates a low-priority node which be used as an arbitrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its `ArbitrationRank` to 1 (the default value) and that of all SQL nodes to 0.

- `ArbitrationDelay`

An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

- `DataDir`

This specifies the directory where output files from the management server will be placed. These files include cluster log files, process output files, and the daemon's process ID (PID) file. (For log files, this location can be overridden by setting the `FILE` parameter for `LogDestination` as discussed previously in this section.)

15.4.4.5. Defining Data Nodes

The `[NDBD]` and `[NDBD DEFAULT]` sections are used to configure the behavior of the cluster's data nodes. There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameters are:

- Either `ExecuteOnComputer` or `HostName`, which must be defined in the local `[NDBD]` section.
- The parameter `NoOfReplicas`, which must be defined in the `[NDBD DEFAULT]` section, as it is common to all Cluster data nodes.

Most data node parameters are set in the `[NDBD DEFAULT]` section. Only those parameters explicitly stated as being able to set local values are allowed to be changed in the `[NDBD]` section. Where present, `HostName`, `Id` and `ExecuteOnComputer` *must* be defined in the local `[NDBD]` section, and not in any other section of `config.ini`. In other words, settings for these parameters are specific to one data node.

For those parameters affecting memory usage or buffer sizes, it is possible to use `K`, `M`, or `G` as a suffix to indicate units of 1024, 1024×1024, or 1024×1024×1024.

(For example, 100K means $100 \times 1024 = 102400$.) Parameter names and values are currently case-sensitive.

Identifying Data Nodes

The `Id` value (that is, the data node identifier) can be allocated on the command line when the node is started or in the configuration file.

- `Id`

This is the node ID used as the address of the node for all cluster internal messages. This is an integer in the range 1 to 63 inclusive. Each node in the cluster must have a unique identity.

- `ExecuteOnComputer`

This refers to the `Id` set for one of the computers defined in a `[COMPUTER]` section.

- `HostName`

Specifying this parameter defines the hostname of the computer on which the data node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `ServerPort` (*OBSOLETE*)

Each node in the cluster uses a port to connect to other nodes. This port is used also for non-TCP transporters in the connection setup phase. The default port is allocated dynamically in such a way as to ensure that no two nodes on the same computer receive the same port number, so it should not normally be necessary to specify a value for this parameter.

- `NoOfReplicas`

This global parameter can be set only in the `[NDBD DEFAULT]` section, and defines the number of replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes all storing the same information.

Node groups are formed implicitly. The first node group is formed by the set of data nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 data nodes and that `NoOfReplicas` is set to 2. The four data nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the same node groups are not placed on the same computer because a single hardware failure would cause the entire cluster to crash.

If no node IDs are provided, the order of the data nodes will be the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client's `SHOW` statement.

There is no default value for `NoOfReplicas`; the maximum possible value is 4.

- `DataDir`

This parameter specifies the directory where trace files, log files, pid files and error logs are placed.

- `FileSystemPath`

This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs and data files are placed. The default is the directory specified by `DataDir`. **Note:** This directory must exist before the **ndbd** process is initiated.

The recommended directory hierarchy for MySQL Cluster includes `/var/lib/mysql-cluster`, under which a directory for the node's filesystem is created. The name of this subdirectory contains the node ID. For example, if the node ID is 2, this subdirectory is named `ndb_2_fs`.

- `BackupDataDir`

This parameter specifies the directory in which backups are placed. If omitted, the default backup location is the directory named `BACKUP` under the location specified by the `FileSystemPath` parameter. (See above.)

Data Memory, Index Memory, and String Memory

DataMemory and IndexMemory are [NDBD] parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to understand how DataMemory and IndexMemory are used, as they usually need to be updated to reflect actual usage by the cluster:

- DataMemory

This parameter defines the amount of space (in bytes) available for storing database records. The entire amount specified by this value is allocated in memory, so it is extremely important that the machine has sufficient physical memory to accommodate it.

The memory allocated by DataMemory is used to store both the actual records and indexes. Each record is currently of fixed size. (Even VARCHAR columns are stored as fixed-width columns.) There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page. The maximum record size is currently 8052 bytes.

The memory space defined by DataMemory is also used to store ordered indexes, which use about 10 bytes per record. Each table row is represented in the ordered index. A common error among users is to assume that all indexes are stored in the memory allocated by IndexMemory, but this is not the case: Only primary key and unique hash indexes use this memory; ordered indexes use the memory allocated by DataMemory. However, creating a primary key or unique hash index also creates an ordered index on the same keys, unless you specify `USING HASH` in the index creation statement. This can be verified by running `ndb_desc -d db_name table_name` in the management client.

The memory space allocated by DataMemory consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are data nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in `NoOfReplicas`.

Once a page has been allocated, it is currently not possible to return it to the

pool of free pages, except by deleting the table. (This also means that DataMemory pages, once allocated to a given table, cannot be used by other tables.) Performing a node recovery also compresses the partition because all records are inserted into empty partitions from other live nodes.

The DataMemory memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the DataMemory. There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure
- Large transactions use more memory

The default value for DataMemory is 80MB; the minimum is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached. This limit is determined by the amount of physical RAM available on the machine and by the amount of memory that the operating system may commit to any one process. 32-bit operating systems are generally limited to 2–4GB per process; 64-bit operating systems can use more. For large databases, it may be preferable to use a 64-bit operating system for this reason. In addition, it is also possible to run more than one **ndbd** process per machine, and this may prove advantageous on machines with multiple CPUs.

- IndexMemory

This parameter controls the amount of storage used for hash indexes in MySQL Cluster. Hash indexes are always used for primary key indexes, unique indexes, and unique constraints. Note that when defining a primary key and a unique index, two indexes will be created, one of which is a hash index used for all tuple accesses as well as lock handling. It is also used to

enforce unique constraints.

The size of the hash index is 25 bytes per record, plus the size of the primary key. For primary keys larger than 32 bytes another 8 bytes is added.

The default value for `IndexMemory` is 18MB. The minimum is 1MB.

- `StringMemory`

This parameter determines how much memory is allocated for strings such as table names, and is specified in an `[NDBD]` or `[NDBD DEFAULT]` section of the `config.ini` file. A value between 0 and 100 inclusive is interpreted as a percent of the maximum default value, which is calculated based on a number of factors including the number of tables, maximum table name size, maximum size of `.FRM` files, `MaxNoOfTriggers`, maximum column name size, and maximum default column value. In general it is safe to assume that the maximum default value is approximately 5 MB for a MySQL Cluster having 1000 tables.

A value greater than 100 is interpreted as a number of bytes.

In MySQL 5.0, the default value is 100 — that is, 100 percent of the default maximum, or roughly 5 MB. It is possible to reduce this value safely, but it should never be less than 5 percent. If you encounter Error 773 Out of string memory, please modify `StringMemory` config parameter: Permanent error: Schema error, this means that means that you have set the `StringMemory` value too low. 25 (25 percent) is not excessive, and should prevent this error from recurring in all but the most extreme conditions, as when there are hundreds or thousands of NDB tables with names whose lengths and columns whose number approach their permitted maximums.

The following example illustrates how memory is used for a table. Consider this table definition:

```
CREATE TABLE example (  
  a INT NOT NULL,  
  b INT NOT NULL,  
  c INT NOT NULL,  
  PRIMARY KEY(a),  
  UNIQUE(b)  
) ENGINE=NDBCLUSTER;
```

For each record, there are 12 bytes of data plus 12 bytes overhead. Having no nullable columns saves 4 bytes of overhead. In addition, we have two ordered indexes on columns a and b consuming roughly 10 bytes each per record. There is a primary key hash index on the base table using roughly 29 bytes per record. The unique constraint is implemented by a separate table with b as primary key and a as a column. This other table consumes an additional 29 bytes of index memory per record in the example table as well 8 bytes of record data plus 12 bytes of overhead.

Thus, for one million records, we need 58MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in MySQL Cluster to handle uniqueness constraints.

Currently, the only partitioning algorithm is hashing and ordered indexes are local to each node. Thus, ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both `IndexMemory` and `DataMemory` is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with two replicas, there will be two node groups. Thus, the total data memory available is $2 \times \text{DataMemory}$ for each data node.

It is highly recommended that `DataMemory` and `IndexMemory` be set to the same values for all nodes. Data distribution is even over all nodes in the cluster, so the maximum amount of space available for any node can be no greater than that of the smallest node in the cluster.

`DataMemory` and `IndexMemory` can be changed, but decreasing either of these can be risky; doing so can easily lead to a node or even an entire MySQL Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be acceptable, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, and then restarting the management server followed by

restarting each data node in turn.

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.

Transaction Parameters

The next three [NDBD] parameters that we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. `MaxNoOfConcurrentTransactions` sets the number of parallel transactions possible in a node. `MaxNoOfConcurrentOperations` sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially `MaxNoOfConcurrentOperations`) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, to ensure that these do not use excessive memory.

- `MaxNoOfConcurrentTransactions`

For each active transaction in the cluster there must be a record in one of the cluster nodes. The task of coordinating transactions is spread among the nodes. The total number of transaction records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

Transaction records are allocated to individual MySQL servers. Normally, there is at least one transaction record allocated per connection that using any table in the cluster. For this reason, one should ensure that there are more transaction records in the cluster than there are concurrent connections to all MySQL servers in the cluster.

This parameter must be set to the same value for all cluster nodes.

Changing this parameter is never safe and doing so can cause a cluster to crash. When a node crashes, one of the nodes (actually the oldest surviving node) will build up the transaction state of all transactions ongoing in the crashed node at the time of the crash. It is thus important that this node has as many transaction records as the failed node.

The default value is 4096.

- `MaxNoOfConcurrentOperations`

It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions of only a few operations each and not involving a great many records, there is no need to set this parameter very high. When performing large transactions involving many records need to set this parameter higher.

Records are kept for each transaction updating cluster data, both in the transaction coordinator and in the nodes where the actual updates are performed. These records contain state information needed to find UNDO records for rollback, lock queues, and other purposes.

This parameter should be set to the number of records to be updated simultaneously in transactions, divided by the number of cluster data nodes. For example, in a cluster which has four data nodes and which is expected to handle 1,000,000 concurrent updates using transactions, you should set this value to $1000000 / 4 = 250000$.

Read queries which set locks also cause operation records to be created. Some extra space is allocated within individual nodes to accommodate cases where the distribution is not perfect over the nodes.

When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

The default value is 32768.

This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction coordinator. The second part specifies how many operation records are to be local to the database.

A very large transaction performed on an eight-node cluster requires as many operation records in the transaction coordinator as there are reads, updates, and deletes involved in the transaction. However, the operation

records of the are spread over all eight nodes. Thus, if it is necessary to configure the system for one very large transaction, it is a good idea to configure the two parts separately. `MaxNoOfConcurrentOperations` will always be used to calculate the number of operation records in the transaction coordinator portion of the node.

It is also important to have an idea of the memory requirements for operation records. These consume about 1KB per record.

- `MaxNoOfLocalOperations`

By default, this parameter is calculated as $1.1 \times \text{MaxNoOfConcurrentOperations}$. This fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, it is a good idea to override the default value by explicitly specifying this parameter.

Transaction Temporary Storage

The next set of [NDBD] parameters is used to determine temporary storage when executing a statement that is part of a Cluster transaction. All records are released when the statement is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these values to enable better parallelism in the system, whereas users whose applications require relatively small transactions can decrease the values to save memory.

- `MaxNoOfConcurrentIndexOperations`

For queries using a unique hash index, another temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus, this record is allocated only while executing a part of a query. As soon as this part has been executed, the record is released. The state needed to handle aborts and commits is handled by the normal operation records, where the pool size is set by the parameter `MaxNoOfConcurrentOperations`.

The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree of parallelism is not required for the cluster.

- `MaxNoOfFiredTriggers`

The default value of `MaxNoOfFiredTriggers` is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short-lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

- `TransactionBufferMemory`

The memory affected by this parameter is used for tracking operations fired when updating index tables and reading unique indexes. This memory is used to store the key and column information for these operations. It is only very rarely that the value for this parameter needs to be altered from the default.

The default value for `TransactionBufferMemory` is 1MB.

Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter `ZATTRBUF_FILESIZE` (found in `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) set to 4000×128 bytes (500KB). A similar buffer for key information, `ZDATABUF_FILESIZE` (also in `Dbtc.hpp`) contains $4000 \times 16 = 62.5$ KB of buffer space. `Dbtc` is the module that handles transaction coordination.

Scans and Buffering

There are additional [NDBD] parameters in the Db1qh module (in `ndb/src/kernel/blocks/Db1qh/Db1qh.hpp`) that affect reads and updates. These include `ZATTRINBUF_FILESIZE`, set by default to 10000×128 bytes (1250KB) and `ZDATABUF_FILE_SIZE`, set by default to 10000×16 bytes (roughly 156KB) of buffer space. To date, there have been neither any reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

- `MaxNoOfConcurrentScans`

This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction coordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain `MaxNoOfConcurrentScans` scans concurrently from all nodes in the cluster.

Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. The order is kept on the local partitions only, so it is necessary to perform the index scan on all partitions.

The default value of `MaxNoOfConcurrentScans` is 256. The maximum value is 500.

This parameter specifies the number of scans possible in the transaction coordinator. If the number of local scan records is not provided, it is calculated as the product of `MaxNoOfConcurrentScans` and the number of data nodes in the system.

- `MaxNoOfLocalScans`

Specifies the number of local scan records if many scans are not fully parallelized.

- BatchSizePerLocalScan

This parameter is used to calculate the number of lock records which must be there to handle many concurrent scan operations.

The default value is 64; this value has a strong connection to the ScanBatchSize defined in the SQL nodes.

- LongMessageBuffer

This is an internal buffer used for passing messages within individual nodes and between nodes. Although it is highly unlikely that this would need to be changed, it is configurable. By default, it is set to 1MB.

Logging and Checkpointing

These [NDBD] parameters control log and checkpoint behavior.

- NoOfFragmentLogFiles

This parameter sets the size of the node's REDO log files. REDO log files are organized in a ring. It is extremely important that the first and last log files (sometimes referred to as the “head” and “tail” log files, respectively) do not meet. When these approach one another too closely, the node begins aborting all transactions encompassing updates due to a lack of room for new log records.

A REDO log record is not removed until three local checkpoints have been completed since that log record was inserted. Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

How these parameters interact and proposals for how to configure them are discussed in [Section 15.4.6, “Configuring Parameters for Local Checkpoints”](#).

The default parameter value is 8, which means 8 sets of 4 16MB files for a total of 512MB. In other words, REDO log space must be allocated in blocks of 64MB. In scenarios requiring a great many updates, the value for NoOfFragmentLogFiles may need to be set as high as 300 or even higher to

provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeopardizing recovery, all updating transactions are aborted with internal error code 410 (Out of log file space temporarily). This condition prevails until a checkpoint has completed and the log tail can be moved forward.

Important: This parameter cannot be changed “on the fly”; you must restart the node using `--initial`. If you wish to change this value for a running cluster, you can do so via a rolling node restart.

- `MaxNoOfSavedMessages`

This parameter sets the maximum number of trace files that are kept before overwriting old ones. Trace files are generated when, for whatever reason, the node crashes.

The default is 25 trace files.

Metadata Objects

The next set of [NDBD] parameters defines pool sizes for metadata objects, used to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters. Note that these act merely as “suggestions” to the cluster, and any that are not specified revert to the default values shown.

- `MaxNoOfAttributes`

Defines the number of attributes that can be defined in the cluster.

The default value is 1000, with the minimum possible value being 32. The maximum is 4294967039. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

When setting `MaxNoOfAttributes`, it is important to prepare in advance for any `ALTER TABLE` statements that you might want to perform in the future. This is due to the fact, during the execution of `ALTER TABLE` on a Cluster

table, 3 times the number of attributes as in the original table are used. For example, if a table requires 100 attributes, and you want to be able to alter it later, you need to set the value of `MaxNoOfAttributes` to 300. Assuming that you can create all desired tables without any problems, a good rule of thumb is to add two times the number of attributes in the largest table to `MaxNoOfAttributes` to be sure. You should also verify that this number is sufficient by trying an actual `ALTER TABLE` after configuring the parameter. If this is not successful, increase `MaxNoOfAttributes` by another multiple of the original value and test it again.

- `MaxNoOfTables`

A table object is allocated for each table, unique hash index, and ordered index. This parameter sets the maximum number of table objects for the cluster as a whole.

For each attribute that has a BLOB data type an extra table is used to store most of the BLOB data. These tables also must be taken into account when defining the total number of tables.

The default value of this parameter is 128. The minimum is 8 and the maximum is 1600. Each table object consumes approximately 20KB per node.

- `MaxNoOfOrderedIndexes`

For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default, each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index.

The default value of this parameter is 128. Each object consumes approximately 10KB of data per node.

- `MaxNoOfUniqueHashIndexes`

For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default, an ordered index is also defined for each unique index. To prevent this, you must specify the `USING HASH` option when defining the unique

index.

The default value is 64. Each index consumes approximately 15KB per node.

- `MaxNoOfTriggers`

Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that three triggers are created for each unique hash index.) However, an *ordered* index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

This parameter sets the maximum number of trigger objects in the cluster.

The default value is 768.

- `MaxNoOfIndexes`

This parameter is deprecated in MySQL 5.0; you should use `MaxNoOfOrderedIndexes` and `MaxNoOfUniqueHashIndexes` instead.

This parameter is used only by unique hash indexes. There needs to be one record in this pool for each unique hash index defined in the cluster.

The default value of this parameter is 128.

Boolean Parameters

The behavior of data nodes is also affected by a set of [NDBD] parameters taking on boolean values. These parameters can each be specified as `TRUE` by setting them equal to 1 or `Y`, and as `FALSE` by setting them equal to 0 or `N`.

- `LockPagesInMainMemory`

For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and so avoid any swapping to disk. This can be used to help guarantee the cluster's real-time characteristics.

This feature is disabled by default.

- `StopOnError`

This parameter specifies whether an **ndbd** process should exit or perform an automatic restart when an error condition is encountered.

This feature is enabled by default.

- `Diskless`

It is possible to specify MySQL Cluster tables as *diskless*, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that neither the tables nor the records in those tables survive a crash. However, when operating in diskless mode, it is possible to run **ndbd** on a diskless computer.

Important: This feature causes the *entire* cluster to operate in diskless mode.

When this feature is enabled, Cluster online backup is disabled. In addition, a partial start of the cluster is not possible.

`Diskless` is disabled by default.

- `RestartOnErrorInsert`

This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

This feature is disabled by default.

Controlling Timeouts, Intervals, and Disk Paging

There are a number of [NDBD] parameters specifying timeouts and intervals between various actions in Cluster data nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this are mentioned where applicable.

- `TimeBetweenWatchDogCheck`

To prevent the main thread from getting stuck in an endless loop at some

point, a “watchdog” thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the process remains in the same state after three checks, the watchdog thread terminates it.

This parameter can easily be changed for purposes of experimentation or to adapt to local conditions. It can be specified on a per-node basis although there seems to be little reason for doing so.

The default timeout is 4000 milliseconds (4 seconds).

- `StartPartialTimeout`

This parameter specifies how long the Cluster waits for all data nodes to come up before the cluster initialization routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

The default value is 30000 milliseconds (30 seconds). 0 disables the timeout. In other words, the cluster may start only if all nodes are available.

- `StartPartitionedTimeout`

If the cluster is ready to start after waiting for `StartPartialTimeout` milliseconds but is still possibly in a partitioned state, the cluster waits until this timeout has also passed.

The default timeout is 60000 milliseconds (60 seconds).

- `StartFailureTimeout`

If a data node has not completed its startup sequence within the time specified by this parameter, the node startup fails. Setting this parameter to 0 means that no data node timeout is applied.

The default value is 60000 milliseconds (60 seconds). For data nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a data node containing several gigabytes of data, a period as long as 10–15 minutes (that is, 600,000 to 1,000,000 milliseconds) might be required to to perform a node restart.

- `HeartbeatIntervalDbDb`

One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. After missing three heartbeat intervals in a row, the node is declared dead. Thus, the maximum time for discovering a failure through the heartbeat mechanism is four times the heartbeat interval.

The default heartbeat interval is 1500 milliseconds (1.5 seconds). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds, obviously the node will be declared dead very quickly. This parameter can be changed during an online software upgrade, but only in small increments.

- `HeartbeatIntervalDbApi`

Each data node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared “dead,” in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until all activities initiated by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for `HeartbeatIntervalDbDb`.

The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between individual data nodes because each data node watches the MySQL servers connected to it, independently of all other data nodes.

- `TimeBetweenLocalCheckpoints`

This parameter is an exception in that it does not specify a time to wait before starting a new local checkpoint; rather, it is used to ensure that local checkpoints are not performed in a cluster where relatively few updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the base-2 logarithm of the number of 4-byte words, so that the

default value 20 means 4MB (4×2^{20}) of write operations, 21 would mean 8MB, and so on up to a maximum value of 31, which equates to 8GB of write operations.

All the write operations in the cluster are added together. Setting `TimeBetweenLocalCheckpoints` to 6 or less means that local checkpoints will be executed continuously without pause, independent of the cluster's workload.

- `TimeBetweenGlobalCheckpoints`

When a transaction is committed, it is committed in main memory in all nodes on which the data is mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behind this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases — a complete crash of the cluster — is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group. Later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- `TimeBetweenInactiveTransactionAbortCheck`

Timeout handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, every transaction will be checked for timing out once per second.

The default value is 1000 milliseconds (1 second).

- `TransactionInactiveTimeout`

This parameter states the maximum time that is permitted to lapse between operations in the same transaction before the transaction is aborted.

The default for this parameter is zero (no timeout). For a real-time database that needs to ensure that no transaction keeps locks for too long, this parameter should be set to a much smaller value. The unit is milliseconds.

- `TransactionDeadlockDetectionTimeout`

When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. A failure to respond can occur for any of the following reasons:

- The node is “dead”
- The operation has entered a lock queue
- The node requested to perform the action could be heavily overloaded.

This timeout parameter states how long the transaction coordinator waits for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection. Setting it too high can cause a undesirable behavior in situations involving deadlocks and node failure.

The default timeout value is 1200 milliseconds (1.2 seconds).

- `NoOfDiskPagesToDiskAfterRestartTUP`

When executing a local checkpoint, the algorithm flushes all data pages to disk. Merely doing so as quickly as possible without any moderation is likely to impose excessive loads on processors, networks, and disks. To control the write speed, this parameter specifies how many pages per 100 milliseconds are to be written. In this context, a “page” is defined as 8KB. This parameter is specified in units of 80KB per second, so , setting `NoOfDiskPagesToDiskAfterRestartTUP` to a value of 20 entails writing 1.6MB in data pages to disk each second during a local checkpoint. This value includes the writing of UNDO log records for data pages. That is, this parameter handles the limitation of writes from data memory. UNDO log records for index pages are handled by the parameter

NoOfDiskPagesToDiskAfterRestartACC. (See the entry for IndexMemory for information about index pages.)

In short, this parameter specifies how quickly to execute local checkpoints. It operates in conjunction with NoOfFragmentLogFiles, DataMemory, and IndexMemory.

For more information about the interaction between these parameters and possible strategies for choosing appropriate values for them, see [Section 15.4.6, “Configuring Parameters for Local Checkpoints”](#).

The default value is 40 (3.2MB of data pages per second).

- NoOfDiskPagesToDiskAfterRestartACC

This parameter uses the same units as NoOfDiskPagesToDiskAfterRestartTUP and acts in a similar fashion, but limits the speed of writing index pages from index memory.

The default value of this parameter is 20 (1.6MB of index memory pages per second).

- NoOfDiskPagesToDiskDuringRestartTUP

This parameter is used in a fashion similar to NoOfDiskPagesToDiskAfterRestartTUP and NoOfDiskPagesToDiskAfterRestartACC, only it does so with regard to local checkpoints executed in the node when a node is restarting. A local checkpoint is always performed as part of all node restarts. During a node restart it is possible to write to disk at a higher speed than at other times, because fewer activities are being performed in the node.

This parameter covers pages written from data memory.

The default value is 40 (3.2MB per second).

- NoOfDiskPagesToDiskDuringRestartACC

Controls the number of index memory pages that can be written to disk during the local checkpoint phase of a node restart.

As with `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC`, values for this parameter are expressed in terms of 8KB pages written per 100 milliseconds (80KB/second).

The default value is 20 (1.6MB per second).

- `ArbitrationTimeout`

This parameter specifies how long data nodes wait for a response from the arbitrator to an arbitration message. If this is exceeded, the network is assumed to have split.

The default value is 1000 milliseconds (1 second).

Buffering and Logging

Several [NDBD] configuration parameters corresponding to former compile-time parameters are also available. These enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front ends to the file system when writing log records to disk. If the node is running in diskless mode, these parameters can be set to their minimum values without penalty due to the fact that disk writes are “faked” by the NDB storage engine's filesystem abstraction layer.

- `UndoIndexBuffer`

The UNDO index buffer, whose size is set by this parameter, is used during local checkpoints. The NDB storage engine uses a recovery scheme based on checkpoint consistency in conjunction with an operational REDO log. To produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

The UNDO index buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the NDB storage engine writes UNDO log records that map all physical changes to an index page so

that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to ensure that these operations need no UNDO logging.

This buffer is 2MB by default. The minimum value is 1MB, which is sufficient for most applications. For applications doing extremely large or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the NDB storage engine issues internal error code 677 (Index UNDO buffers overloaded).

Important: It is not safe to decrease the value of this parameter during a rolling restart.

- UndoDataBuffer

This parameter sets the size of the UNDO data buffer, which performs a function similar to that of the UNDO index buffer, except the UNDO data buffer is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.

Because UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

This amount of memory may be unnecessarily large for some applications. In such cases, it is possible to decrease this size to a minimum of 1MB.

It is rarely necessary to increase the size of this buffer. If there is such a need, it is a good idea to check whether the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

If this buffer is too small and gets congested, the NDB storage engine issues internal error code 891 (Data UNDO buffers overloaded).

Important: It is not safe to decrease the value of this parameter during a rolling restart.

- RedoBuffer

All update activities also need to be logged. The REDO log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a “fuzzy” checkpoint of the data together with the UNDO log, and then applies the REDO log to play back all changes up to the restoration point.

RedoBuffer sets the size of the buffer in which the REDO log is written, and is 8MB by default. The minimum value is 1MB.

If this buffer is too small, the NDB storage engine issues error code 1221 (REDO log buffers overloaded).

Important: It is not safe to decrease the value of this parameter during a rolling restart.

Controlling Log Messages

In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to stdout. For each event category, there are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to stdout; setting it to 0 means that there will be no event reports made in that category.

By default, only the startup message is sent to stdout, with the remaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server's cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

- LogLevelStartup

The reporting level for events generated during startup of the process.

The default level is 1.

- `LogLevelShutdown`

The reporting level for events generated as part of graceful shutdown of a node.

The default level is 0.

- `LogLevelStatistic`

The reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.

The default level is 0.

- `LogLevelCheckpoint`

The reporting level for events generated by local and global checkpoints.

The default level is 0.

- `LogLevelNodeRestart`

The reporting level for events generated during node restart.

The default level is 0.

- `LogLevelConnection`

The reporting level for events generated by connections between cluster nodes.

The default level is 0.

- `LogLevelError`

The reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

The default level is 0.

- `LogLevelInfo`

The reporting level for events generated for information about the general state of the cluster.

The default level is 0.

Backup Parameters

The [NDBD] parameters discussed in this section define memory buffers set aside for execution of online backups.

- `BackupDataBufferSize`

In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as `BackupwriteSize` (see below), the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens, the backup process pauses the scan and waits until some disk writes have completed freed up memory so that scanning may continue.

The default value is 2MB.

- `BackupLogBufferSize`

The backup log buffer fulfills a role similar to that played by the backup data buffer, except that it is used for generating a log of all table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the backup log buffer must be large enough to handle the load caused by write activities while the backup is being made. See [Section 15.8.4, “Configuration for Cluster Backup”](#).

The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by

insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster is unlikely to be able to perform the desired operations.

It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

The default value is 2MB.

- BackupMemory

This parameter is simply the sum of BackupDataBufferSize and BackupLogBufferSize.

The default value is $2\text{MB} + 2\text{MB} = 4\text{MB}$.

Important: If BackupDataBufferSize and BackupLogBufferSize taken together exceed 4MB, then this parameter must be set explicitly in the config.ini file to their sum.

- BackupWriteSize

This parameter specifies the size of messages written to disk by the backup log and backup data buffers.

The default value is 32KB.

15.4.4.6. Defining SQL Nodes

The [MYSQLD] sections in the config.ini file define the behavior of the MySQL servers (SQL nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, any host can use this SQL node.

- Id

The Id value is used to identify the node in all cluster internal messages. It must be an integer in the range 1 to 63 inclusive, and must be unique among

all node IDs within the cluster.

- `ExecuteOnComputer`

This refers to the `Id` set for one of the computers (hosts) defined in a `[COMPUTER]` section of the configuration file.

- `HostName`

Specifying this parameter defines the hostname of the computer on which the SQL node (API node) is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `ArbitrationRank`

This parameter defines which nodes can act as arbitrators. Both MGM nodes and SQL nodes can be arbitrators. A value of 0 means that the given node is never used as an arbitrator, a value of 1 gives the node high priority as an arbitrator, and a value of 2 gives it low priority. A normal configuration uses the management server as arbitrator, setting its `ArbitrationRank` to 1 (the default) and those for all SQL nodes to 0.

- `ArbitrationDelay`

Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests will be delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- `BatchByteSize`

For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records (`BatchSize`) and in terms of bytes (`BatchByteSize`). The actual batch size is limited by both parameters.

The speed at which queries are performed can vary by more than 40% depending upon how this parameter is set. In future releases, MySQL Server will make educated guesses on how to set parameters relating to

batch size, based on the query type.

This parameter is measured in bytes and by default is equal to 32KB.

- BatchSize

This parameter is measured in number of records and is by default set to 64. The maximum size is 992.

- MaxScanBatchSize

The batch size is the size of each batch sent from each data node. Most scans are performed in parallel to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

The default value of this parameter is set to 256KB. Its maximum size is 16MB.

You can obtain some information from a MySQL server running as a Cluster SQL node using `SHOW STATUS` in the `mysql` client, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----+-----+
| Variable_name          | Value          |
+-----+-----+
| Ndb_cluster_node_id   | 5              |
| Ndb_config_from_host   | 192.168.0.112 |
| Ndb_config_from_port   | 1186           |
| Ndb_number_of_storage_nodes | 4              |
+-----+-----+
4 rows in set (0.02 sec)
```

For information about these Cluster system status variables, see [Section 5.2.4, “Server Status Variables”](#).

15.4.4.7. Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for establishing connections in MySQL Cluster. It is normally not necessary to define connections because Cluster automatically set ups a connection between each of the data nodes, between each data node and all MySQL server nodes, and between each data

node and the management server. (For one exception to this rule, see [Section 15.4.4.8, “TCP/IP Connections Using Direct Connections”](#).) [TCP] sections in the `config.ini` file explicitly define TCP/IP connections between nodes in the cluster.

It is only necessary to define a connection to override the default connection parameters. In that case, it is necessary to define at least `NodeId1`, `NodeId2`, and the parameters to change.

It is also possible to change the default values for these parameters by setting them in the [TCP DEFAULT] section.

- `NodeId1`, `NodeId2`

To identify a connection between two nodes it is necessary to provide their node IDs in the [TCP] section of the configuration file. These are the same unique `Id` values for each of these nodes as described in [Section 15.4.4.6, “Defining SQL Nodes”](#).

- `SendBufferMemory`

TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these are also sent when a round of messages have been executed. To handle temporary overload situations it is also possible to define a bigger send buffer. The default size of the send buffer is 256KB.

- `SendSignalId`

To be able to retrace a distributed message datagram, it is necessary to identify each message. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default.

- `Checksum`

This parameter is a boolean parameter (enabled by setting it to `Y` or `1`, disabled by setting it to `N` or `0`). It is disabled by default. When it is enabled, checksums for all messages are calculated before they are placed in the send buffer. This feature ensures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- PortNumber (*OBSOLETE*)

This formerly specified the port number to be used for listening for connections from other nodes. This parameter should no longer be used.

- ReceiveBufferMemory

Specifies the size of the buffer used when receiving data from the TCP/IP socket. There is seldom any need to change this parameter from its default value of 64KB, except possibly to save memory.

15.4.4.8. TCP/IP Connections Using Direct Connections

Setting up a cluster using direct connections between data nodes requires specifying explicitly the crossover IP addresses of the data nodes so connected in the [TCP] section of the cluster config.ini file.

In the following example, we envision a cluster with at least four hosts, one each for a management server, an SQL node, and two data nodes. The cluster as a whole resides on the 172.23.72.* subnet of a LAN. In addition to the usual network connections, the two data nodes are connected directly using a standard crossover cable, and communicate with one another directly using IP addresses in the 1.1.0.* address range as shown:

```
# Management Server
[NDB_MGMD]
Id=1
HostName=172.23.72.20

# SQL Node
[MYSQLD]
Id=2
HostName=172.23.72.21

# Data Nodes
[NDBD]
Id=3
HostName=172.23.72.22

[NDBD]
Id=4
HostName=172.23.72.23
```

```
# TCP/IP Connections
[TCP]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

The `HostName N` parameter, where N is an integer, is used only when specifying direct TCP/IP connections.

The use of direct connections between data nodes can improve the cluster's overall efficiency by allowing the data nodes to bypass an Ethernet device such as a switch, hub, or router, thus cutting down on the cluster's latency. It is important to note that to take the best advantage of direct connections in this fashion with more than two data nodes, you must have a direct connection between each data node and every other data node in the same node group.

15.4.4.9. Shared-Memory Connections

MySQL Cluster attempts to use the shared memory transporter and configure it automatically where possible, chiefly where more than one node runs concurrently on the same cluster host. (In very early versions of MySQL Cluster, shared memory segments functioned only when the server binary was built using `--with-ndb-shm`.) [SHM] sections in the `config.ini` file explicitly define shared-memory connections between nodes in the cluster. When explicitly defining shared memory as the connection method, it is necessary to define at least `NodeId1`, `NodeId2` and `ShmKey`. All other parameters have default values that should work well in most cases.

Important: *SHM functionality is considered experimental only.* It is not officially supported in any MySQL release series up to and including 5.0. This means that you must determine for yourself or by using our free resources (forums, mailing lists) whether it can be made to work correctly in your specific case.

- `NodeId1`, `NodeId2`

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- ShmKey

When setting up shared memory segments, a node ID, expressed as an integer, is used to identify uniquely the shared memory segment to use for the communication. There is no default value.

- ShmSize

Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by `ShmSize`. The default value is 1MB.

- SendSignalId

To retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to `Y` causes these message IDs to be transported over the network as well. This feature is disabled by default.

- Checksum

This parameter is a boolean (`Y/N`) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before being placed in the send buffer.

This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

15.4.4.10. SCI Transport Connections

[SCI] sections in the `config.ini` file explicitly define SCI (Scalable Coherent Interface) connections between cluster nodes. Using SCI transporters in MySQL Cluster is supported only when the MySQL-Max binaries are built using `--with-ndb-sci=/your/path/to/SCI`. The *path* should point to a directory that contains at a minimum `lib` and `include` directories containing SISC libraries and header files. (See [Section 15.9, “Using High-Speed Interconnects with MySQL Cluster”](#) for more information about SCI.)

In addition, SCI requires specialized hardware.

It is strongly recommended to use SCI Transporters only for communication between **ndbd** processes. Note also that using SCI Transporters means that the **ndbd** processes never sleep. For this reason, SCI Transporters should be used only on machines having at least two CPUs dedicated for use by **ndbd** processes. There should be at least one CPU per **ndbd** process, with at least one CPU left in reserve to handle operating system activities.

- `NodeId1, NodeId2`

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `Host1SciId0`

This identifies the SCI node ID on the first Cluster node (identified by `NodeId1`).

- `Host1SciId1`

It is possible to set up SCI Transporters for failover between two SCI cards which then should use separate networks between the nodes. This identifies the node ID and the second SCI card to be used on the first node.

- `Host2SciId0`

This identifies the SCI node ID on the second Cluster node (identified by `NodeId2`).

- `Host2SciId1`

When using two SCI cards to provide failover, this parameter identifies the second SCI card to be used on the second node.

- `SharedBufferSize`

Each SCI transporter has a shared memory segment used for communication between the two nodes. Setting the size of this segment to the default value of 1MB should be sufficient for most applications. Using a

smaller value can lead to problems when performing many parallel inserts; if the shared buffer is too small, this can also result in a crash of the **ndbd** process.

- **SendLimit**

A small buffer in front of the SCI media stores messages before transmitting them over the SCI network. By default, this is set to 8KB. Our benchmarks show that performance is best at 64KB but 16KB reaches within a few percent of this, and there was little if any advantage to increasing it beyond 8KB.

- **SendSignalId**

To trace a distributed message it is necessary to identify each message uniquely. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default.

- **Checksum**

This parameter is a boolean value, and is disabled by default. When **Checksum** is enabled, checksums are calculated for all messages before they are placed in the send buffer. This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

15.4.5. Overview of Cluster Configuration Parameters

The next three sections provide summary tables of MySQL Cluster configuration parameters used in the `config.ini` file to govern the cluster's functioning. Each table lists the parameters for one of the Cluster node process types (**ndbd**, **ndb_mgmd**, and **mysqld**), and includes the parameter's type as well as its default, minimum, and maximum values as applicable.

It is also stated what type of restart is required (node restart or system restart) — and whether the restart must be done with `--initial` — to change the value of a given configuration parameter. This information is provided in each table's **Restart Type** column, which contains one of the values shown in this list:

- **N**: Node Restart

- **IN:** Initial Node Restart
- **S:** System Restart
- **IS:** Initial System Restart

When performing a node restart or an initial node restart, all of the cluster's data nodes must be restarted in turn (also referred to as a *rolling restart*). It is possible to update cluster configuration parameters marked **N** or **IN** online — that is, without shutting down the cluster — in this fashion. An initial node restart requires restarting each **ndbd** process with the `--initial` option.

A system restart requires a complete shutdown and restart of the entire cluster. An initial system restart requires taking a backup of the cluster, wiping the cluster filesystem after shutdown, and then restoring from the backup following the restart.

In any cluster restart, all of the cluster's management servers must be restarted in order for them to read the updated configuration parameter values.

Important: Values for numeric cluster parameters can generally be increased without any problems, although it is advisable to do so progressively, making such adjustments in relatively small increments. However, decreasing the values of such parameters — particularly those relating to memory usage and disk space — is not to be undertaken lightly, and it is recommended that you do so only following careful planning and testing. In addition, it is the generally the case that parameters relating to memory and disk usage which can be raised using a simple node restart require an initial node restart to be lowered.

Because some of these parameters can be used for configuring more than one type of cluster node, they may appear in more than one of the tables.

(Note that 4294967039 — which often appears as a maximum value in these tables — is equal to $2^{32} - 2^8 - 1$.)

15.4.5.1. Data Node Configuration Parameters

The following table provides information about parameters used in the `[NDBD]` or `[NDB_DEFAULT]` sections of a `config.ini` file for configuring MySQL Cluster

data nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 15.4.4.5, “Defining Data Nodes”](#).

Restart Type Column Values

- N: Node Restart
- IN: Initial Node Restart
- S: System Restart
- IS: Initial System Restart

See [Section 15.4.5, “Overview of Cluster Configuration Parameters”](#), for additional explanations of these abbreviations.

Parameter Name	Type/Units	Default Value
ArbitrationTimeout	milliseconds	1000
BackupDataBufferSize	bytes	2M
BackupDataDir	string	FileSystemPath/
BackupLogBufferSize	bytes	2M
BackupMemory	bytes	4M
BackupWriteSize	bytes	32K
BatchSizePerLocalScan	integer	64
DataDir	string	/var/lib/mysql-
DataMemory	bytes	80M
Diskless	true false (1 0)	0
ExecuteOnComputer	integer	
FileSystemPath	string	value specified fo

HeartbeatIntervalDbApi	milliseconds	1500
HeartbeatIntervalDbDb	milliseconds	1500
HostName	string	localhost
Id	integer	<i>None</i>
IndexMemory	bytes	18M
LockPagesInMainMemory	true false (1 0)	0
LogLevelCheckpoint	integer	0
LogLevelConnection	integer	0
LogLevelError	integer	0
LogLevelInfo	integer	0
LogLevelNodeRestart	integer	0
LogLevelShutdown	integer	0
LogLevelStartup	integer	1
LogLevelStatistic	integer	0
LongMessageBuffer	bytes	1M
MaxNoOfAttributes	integer	1000
MaxNoOfConcurrentIndexOperations	integer	8K
MaxNoOfConcurrentOperations	integer	32768
MaxNoOfConcurrentScans	integer	256
MaxNoOfConcurrentTransactions	integer	4096
MaxNoOfFiredTriggers	integer	4000
MaxNoOfIndexes (<i>DEPRECATED</i> — use MaxNoOfOrderedIndexes or MaxNoOfUniqueHashIndexes instead)	integer	128
MaxNoOfLocalOperations	integer	UNDEFINED
MaxNoOfLocalScans	integer	UNDEFINED
MaxNoOfOrderedIndexes		

	integer	128
MaxNoOfSavedMessages	integer	25
MaxNoOfTables	integer	128
MaxNoOfTriggers	integer	768
MaxNoOfUniqueHashIndexes	integer	64
NoOfDiskPagesToDiskAfterRestartACC	integer (number of 8KB pages per 100 milliseconds)	20 (= 20 * 80KB 1.6MB/second)
NoOfDiskPagesToDiskAfterRestartTUP	integer (number of 8KB pages per 100 milliseconds)	40 (= 40 * 80KB 3.2MB/second)
NoOfDiskPagesToDiskDuringRestartACC	integer (number of 8KB pages per 100 milliseconds)	20 (= 20 * 80KB 1.6MB/second)
NoOfDiskPagesToDiskDuringRestartTUP	integer (number of 8KB pages per 100 milliseconds)	40 (= 40 * 80KB 3.2MB/second)
NoOfFragmentLogFiles	integer	8
NoOfReplicas	integer	<i>None</i>
RedoBuffer	bytes	8M
RestartOnErrorInsert (<i>DEBUG BUILDS ONLY</i>)	true false (1 0)	0
ServerPort (<i>OBSOLETE</i>)	integer	1186
StartFailureTimeout	milliseconds	60000
StartPartialTimeout	milliseconds	30000
StartPartitionedTimeout	milliseconds	60000

StopOnError	true false (1 0)	1
TimeBetweenGlobalCheckpoints	milliseconds	2000
TimeBetweenInactiveTransactionAbortCheck	milliseconds	1000
TimeBetweenLocalCheckpoints	integer (number of 4-byte words as a base-2 logarithm)	20 (= 4 * 2 ²⁰ = 4 operations)
TimeBetweenWatchDogCheck	milliseconds	4000
TransactionBufferMemory	bytes	1M
TransactionDeadlockDetectionTimeout	milliseconds	1200
TransactionInactiveTimeout	milliseconds	0
UndoDataBuffer	bytes	16M
UndoIndexBuffer	bytes	2M

15.4.5.2. Management Node Configuration Parameters

The following table provides information about parameters used in the [NDB_MGMD] or [MGM] sections of a config.ini file for configuring MySQL Cluster management nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 15.4.4.4, “Defining the Management Server”](#).

Restart Type Column Values

- N: Node Restart
- IN: Initial Node Restart
- S: System Restart
- IS: Initial System Restart

See [Section 15.4.5, “Overview of Cluster Configuration Parameters”](#), for additional explanations of these abbreviations.

Parameter Name	Type/Units	Default Value	Minimum Value	Maximum Value	Restart Type
ArbitrationDelay	milliseconds	0	0	4294967039	N
ArbitrationRank	integer	1	0	2	N
DataDir	string	N/A	N/A	N/A	IN
ExecuteOnComputer	integer				
HostName	string	localhost	N/A	N/A	IN
Id	integer	<i>None</i>	1	63	IN
LogDestination	CONSOLE, SYSLOG, or FILE	CONSOLE	N/A	N/A	N

15.4.5.3. SQL Node Configuration Parameters

The following table provides information about parameters used in the [API] sections of a `config.ini` file for configuring MySQL Cluster SQL nodes. For detailed descriptions and other additional information about each of these parameters, see [Section 15.4.4.6, “Defining SQL Nodes”](#).

Restart Type Column Values

- N: Node Restart
- IN: Initial Node Restart
- S: System Restart
- IS: Initial System Restart

See [Section 15.4.5, “Overview of Cluster Configuration Parameters”](#), for additional explanations of these abbreviations.

Parameter Name	Type/Units	Default Value	Minimum Value	Maximum Value	Restart Type
ArbitrationDelay	milliseconds	0	0	4294967039	N
ArbitrationRank	integer	1	0	2	N

BatchByteSize	bytes	32K	1K	1M	N
BatchSize	integer	64	1	992	N
ExecuteOnComputer	integer				
HostName	string	localhost	N/A	N/A	IN
Id	integer	<i>None</i>	1	63	IN
MaxScanBatchSize	bytes	256K	32K	16M	N

15.4.6. Configuring Parameters for Local Checkpoints

The parameters discussed in [Logging and Checkpointing](#) and in [Data Memory, Index Memory, and String Memory](#) that are used to configure local checkpoints for a MySQL Cluster do not exist in isolation, but rather are very much interdependent on each other. In this section, we illustrate how these parameters — including `DataMemory`, `IndexMemory`, `NoOfDiskPagesToDiskAfterRestartTUP`, `NoOfDiskPagesToDiskAfterRestartACC`, and `NoOfFragmentLogFiles` — relate to one another in a working Cluster.

In this example, we assume that our application performs the following numbers of types of operations per hour:

- 50000 selects
- 15000 inserts
- 15000 updates
- 15000 deletes

We also make the following assumptions about the data used in the application:

- We are working with a single table having 40 columns.
- Each column can hold up to 32 bytes of data.
- A typical UPDATE run by the application affects the values of 5 columns.
- No NULL values are inserted by the application.

A good starting point is to determine the amount of time that should elapse between local checkpoints (LCPs). It worth noting that, in the event of a system restart, it takes 40-60 percent of this interval to execute the REDO log — for example, if the time between LCPs is 5 minutes (300 seconds), then it should take 2 to 3 minutes (120 to 180 seconds) for the REDO log to be read.

The maximum amount of data per node can be assumed to be the size of the `DataMemory` parameter. In this example, we assume that this is 2 GB. The `NoOfDiskPagesToDiskAfterRestartTUP` parameter represents the amount of data to be checkpointed per unit time — however, this parameter is actually expressed as the number of 8K memory pages to be checkpointed per 100 milliseconds. 2 GB per 300 seconds is approximately 6.8 MB per second, or 700 KB per 100 milliseconds, which works out to roughly 85 pages per 100 milliseconds.

Similarly, we can calculate `NoOfDiskPagesToDiskAfterRestartACC` in terms of the time for local checkpoints and the amount of memory required for indexes — that is, the `IndexMemory`. Assuming that we allow 512 MB for indexes, this works out to approximately 20 8-KB pages per 100 milliseconds for this parameter.

Next, we need to determine the number of REDO logfiles required — that is, fragment log files — the corresponding parameter being `NoOfFragmentLogFiles`. We need to make sure that there are sufficient REDO logfiles for keeping records for at least 3 local checkpoints. In a production setting, there are always uncertainties — for instance, we cannot be sure that disks always operate at top speed or with maximum throughput. For this reason, it is best to err on the side of caution, so we double our requirement and calculate a number of fragment logfiles which should be enough to keep records covering 6 local checkpoints.

It is also important to remember that the disk also handles writes to the REDO log and UNDO log, so if you find that the amount of data being written to disk as determined by the values of `NoOfDiskPagesToDiskAfterRestartACC` and `NoOfDiskPagesToDiskAfterRestartTUP` is approaching the amount of disk bandwidth available, you may wish to increase the time between local checkpoints.

Given 5 minutes (300 seconds) per local checkpoint, this means that we need to

support writing log records at maximum speed for $6 * 300 = 1800$ seconds. The size of a REDO log record is 72 bytes plus 4 bytes per updated column value plus the maximum size of the updated column, and there is one REDO log record for each table record updated in a transaction, on each node where the data reside. Using the numbers of operations set out previously in this section, we derive the following:

- 50000 select operations per hour yields 0 log records (and thus 0 bytes), since SELECT statements are not recorded in the REDO log.
- 15000 DELETE statements per hour is approximately 5 delete operations per second. (Since we wish to be conservative in our estimate, we round up here and in the following calculations.) No columns are updated by deletes, so these statements consume only 5 operations * 72 bytes per operation = 360 bytes per second.
- 15000 UPDATE statements per hour is roughly the same as 5 updates per second. Each update uses 72 bytes, plus 4 bytes per column * 5 columns updated, plus 32 bytes per column * 5 columns — this works out to $72 + 20 + 160 = 252$ bytes per operation, and multiplying this by 5 operation per second yields 1260 bytes per second.
- 15000 INSERT statements per hour is equivalent to 5 insert operations per second. Each insert requires REDO log space of 72 bytes, plus 4 bytes per record * 40 columns, plus 32 bytes per column * 40 columns, which is $72 + 160 + 1280 = 1512$ bytes per operation. This times 5 operations per second yields 7560 bytes per second.

So the total number of REDO log bytes being written per second is approximately $0 + 360 + 1260 + 7560 = 9180$ bytes. Multiplied by 1800 seconds, this yields 16524000 bytes required for REDO logging, or approximately 15.75 MB. The unit used for `NoOfFragmentLogFiles` represents a set of 4 16-MB logfiles — that is, 64 MB. Thus, the minimum value (3) for this parameter is sufficient for the scenario envisioned in this example, since 3 times 64 = 192 MB, or about 12 times what is required; the default value of 8 (or 512 MB) is more than ample in this case.

A copy of each altered table record is kept in the UNDO log. In the scenario discussed above, the UNDO log would not require any more space than what is

provided by the default settings. However, given the size of disks, it is sensible to allocate at least 1 GB for it.

15.5. Upgrading and Downgrading MySQL Cluster

This portion of the MySQL Cluster chapter covers upgrading and downgrading a MySQL Cluster from one MySQL release to another. It discusses different types of Cluster upgrades and downgrades, and provides a Cluster upgrade/downgrade compatibility matrix (see [Section 15.5.2, “Cluster Upgrade and Downgrade Compatibility”](#)).

Important: You are expected already to be familiar with installing and configuring a MySQL Cluster prior to attempting an upgrade or downgrade. See [Section 15.4, “MySQL Cluster Configuration”](#).

This section remains in development, and will be updated and expanded considerably during the second quarter of 2006.

15.5.1. Performing a Rolling Restart of the Cluster

This section discusses how to perform a *rolling restart* of a MySQL Cluster installation, so called because it involves stopping and starting (or restarting) each node in turn, so that the cluster itself remains operational. This is often done as part of a *rolling upgrade* or *rolling downgrade*, where high availability of the cluster is mandatory and no downtime of the cluster as a whole is permissible. Where we refer to upgrades, the information provided here also generally applies to downgrades as well.

There are a number of reasons why a rolling restart might be desirable:

- **Cluster Configuration Change:** To make a change in the cluster's configuration, such as adding an SQL node to the cluster, or setting a configuration parameter to a new value.
- **Cluster Software Upgrade/Downgrade:** To upgrade the cluster to a newer version of the MySQL Cluster software (or to downgrade it to an older version). This is usually referred to as a “rolling upgrade” (or “rolling downgrade”, when reverting to an older version of MySQL Cluster).
- **Change on Node Host:** To make changes in the hardware or operating system on which one or more cluster nodes are running

- **Cluster Reset:** To reset the cluster because it has reached an undesirable state

The process for performing a rolling restart may be generalised as follows:

1. Stop, reconfigure, then restart each cluster management node (**ndb_mgmd** process) in turn
2. Stop, reconfigure, then restart each cluster data node (**ndbd** process) in turn
3. Stop, reconfigure, then restart each cluster SQL node (**mysqld** process) in turn

The specifics for implementing a particular rolling upgrade depend upon the actual changes being made. A more detailed view of the process is presented in the table shown here:

Table 15.1. Steps for Cluster rolling restarts — by type
RESTART TYPE:

Cluster Configuration Change	Cluster Software Upgrade/Downgrade	Change on Node Host	Cluster
A. For each ndb_mgmd process...			
1. Stop ndb_mgmd	1. Stop ndb_mgmd	1. Stop ndb_mgmd	
2. Make change in configuration file	2. Replace ndb_mgmd binary with new version	2. Make desired changes in hardware/operating system	1. Stop ndb_r
3. Start ndb_mgmd	3. Start ndb_mgmd	3. Start ndb_mgmd	2. Start ndb_r
B. For each ndbd process...			
(OR)			
	1. Stop ndbd	1. Stop ndbd	

- | | | | | |
|----------------------|-------------|--|--|------------|
| 1. Stop ndbd | | 2. Replace ndbd binary with new version | 2. Make desired changes in hardware/operating system | 1. Stop n |
| | Restart | | | |
| 2. Start ndbd | ndbd | 3. Start ndbd | 3. Start ndbd | 2. Start r |

C. For each **mysqld** process...

(OR)

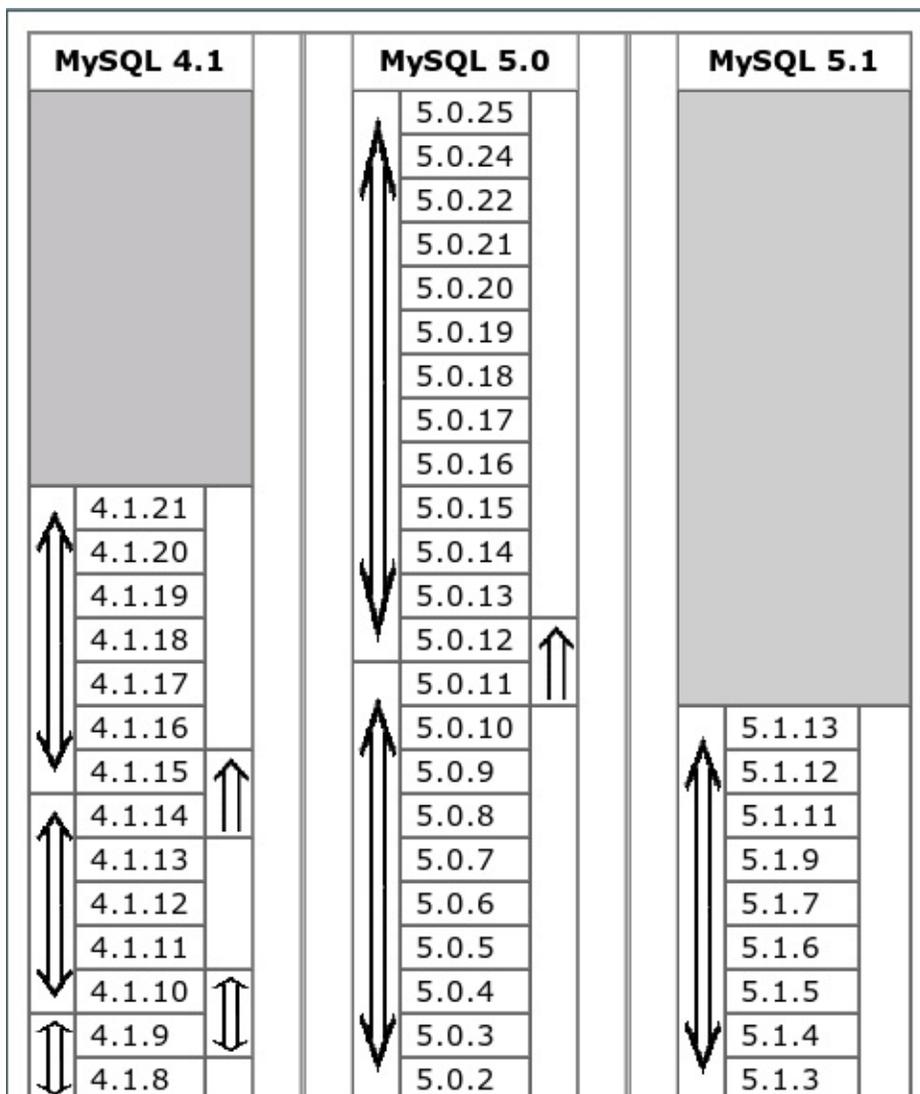
- | | | | | |
|------------------------|---------------|--|--|-----------------------|
| | | 1. Stop mysqld | 1. Stop mysqld | |
| 1. Stop mysqld | Restart | 2. Replace mysqld binary with new version | 2. Make desired changes in hardware/operating system | 1. Stop mysql |
| | mysqld | | | |
| 2. Start mysqld | | 3. Start mysqld | 3. Start mysqld | 2. Start mysql |

15.5.2. Cluster Upgrade and Downgrade Compatibility

This section provides information regarding Cluster software and table file compatibility between differing versions of the MySQL Server for purposes of performing upgrades and downgrades.

Important: Only compatibility between MySQL versions with regard to NDB Cluster is taken into account in this section, and there are likely other issues to be considered. *As with any other MySQL software upgrade or downgrade, you are strongly encouraged to review the relevant portions of the MySQL Manual for the MySQL versions from which and to which you intend to migrate, before attempting an upgrade or downgrade of the MySQL Cluster software. See [Section 2.11, “Upgrading MySQL”](#).*

The following table shows Cluster upgrade and downgrade compatibility between different versions of the MySQL Server.



downgrade from 4.1.10 to 4.1.9, then downgrade from 4.1.9 to 4.1.8.

If you wish to upgrade a MySQL Cluster to 4.1.15, you must upgrade to 4.1.14 first, and you must upgrade to 4.1.15 before upgrading to 4.1.16 or newer.

Cluster downgrades from 4.1.15 to 4.1.14 (or earlier versions) are not supported.

Cluster upgrades from MySQL Server versions previous to 4.1.8 are not supported; when upgrading from these, you must dump all NDB tables, install the new version of the software, and then reload the tables from the dump.

- **5.0 Series:**

MySQL 5.0.2 was the first public release in this series.

Cluster downgrades from MySQL 5.0 to MySQL 4.1 are not supported.

Cluster downgrades from 5.0.12 to 5.0.11 (or earlier) are not supported.

You cannot restore with **ndb_restore** to a MySQL 5.0 Cluster using a backup made from a Cluster running MySQL 5.1. You must use **mysqldump** in such cases.

There was no public release for MySQL 5.0.23.

- **5.1 Series:**

MySQL 5.1.3 was the first public release in this series.

You cannot downgrade a MySQL 5.1.6 or later Cluster using Disk Data tables to MySQL 5.1.5 or earlier unless you convert all such tables to in-memory Cluster tables first.

MySQL 5.1.8 and MySQL 5.1.10 were not released.

15.6. Process Management in MySQL Cluster

Understanding how to manage MySQL Cluster requires a knowledge of four essential processes. In the next few sections of this chapter, we cover the roles played by these processes in a cluster, how to use them, and what startup options are available for each of them:

- [Section 15.6.1, “MySQL Server Process Usage for MySQL Cluster”](#)
- [Section 15.6.2, “ndbd, the Storage Engine Node Process”](#)
- [Section 15.6.3, “ndb mgmd, the Management Server Process”](#)
- [Section 15.6.4, “ndb mgm, the Management Client Process”](#)

15.6.1. MySQL Server Process Usage for MySQL Cluster

mysqld is the traditional MySQL server process. To be used with MySQL Cluster, **mysqld** needs to be built with support for the NDB Cluster storage engine, as it is in the precompiled `-max` binaries available from <http://dev.mysql.com/downloads/>. If you build MySQL from source, you must invoke **configure** with the `--with-ndbcluster` option to enable NDB Cluster storage engine support.

If the **mysqld** binary has been built with Cluster support, the NDB Cluster storage engine is still disabled by default. You can use either of two possible options to enable this engine:

- Use `--ndbcluster` as a startup option on the command line when starting **mysqld**.
- Insert a line containing `ndbcluster` in the `[mysqld]` section of your `my.cnf` file.

An easy way to verify that your server is running with the NDB Cluster storage engine enabled is to issue the `SHOW ENGINES` statement in the MySQL Monitor (**mysql**). You should see the value `YES` as the `Support` value in the row for `NDBCLUSTER`. If you see `NO` in this row or if there is no such row displayed in the

output, you are not running an NDB-enabled version of MySQL. If you see `DISABLED` in this row, you need to enable it in either one of the two ways just described.

To read cluster configuration data, the MySQL server requires at a minimum three pieces of information:

- The MySQL server's own cluster node ID
- The hostname or IP address for the management server (MGM node)
- The number of the TCP/IP port on which it can connect to the management server

Node IDs can be allocated dynamically, so it is not strictly necessary to specify them explicitly.

The **mysqld** parameter `ndb-connectstring` is used to specify the connectstring either on the command line when starting **mysqld** or in `my.cnf`. The connectstring contains the hostname or IP address where the management server can be found, as well as the TCP/IP port it uses.

In the following example, `ndb_mgmd.mysql.com` is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
shell> mysqld --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See [Section 15.4.4.2, “The Cluster connectstring”](#), for more information on connectstrings.

Given this information, the MySQL server will be a full participant in the cluster. (We sometimes refer to a **mysqld** process running in this manner as an SQL node.) It will be fully aware of all cluster data nodes as well as their status, and will establish connections to all data nodes. In this case, it is able to use any data node as a transaction coordinator and to read and update node data.

You can see in the **mysql** client whether a MySQL server is connected to the cluster using `SHOW PROCESSLIST`. If the MySQL server is connected to the cluster, and you have the `PROCESS` privilege, then the first row of the output is as

shown here:

```
mysql> SHOW PROCESSLIST \G
***** 1. row *****
  Id: 1
  User: system user
  Host:
  db:
Command: Daemon
  Time: 1
  State: Waiting for event from ndbcluster
  Info: NULL
```

15.6.2. ndbd, the Storage Engine Node Process

ndbd is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process that empowers a data node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In a MySQL Cluster, a set of **ndbd** processes cooperate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between data nodes and Cluster hosts is completely configurable.

ndbd generates a set of log files which are placed in the directory specified by `DataDir` in the `config.ini` configuration file. These log files are listed below. Note that *node_id* represents the node's unique identifier. For example, `ndb_2_error.log` is the error log generated by the data node whose node ID is 2.

- `ndb_node_id_error.log` is a file containing records of all crashes which the referenced **ndbd** process has encountered. Each record in this file contains a brief error string and a reference to a trace file for this crash. A typical entry in this file might appear as shown here:

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
```

```
TraceFile: ndb_2_trace.log.2
***EOM***
```

Note: *It is very important to be aware that the last entry in the error log file is not necessarily the newest one (nor is it likely to be). Entries in the error log are not listed in chronological order; rather, they correspond to the order of the trace files as determined in the ndb_node_id_trace.log.next file (see below). Error log entries are thus overwritten in a cyclical and not sequential fashion.*

- `ndb_node_id_trace.log.trace_id` is a trace file describing exactly what happened just before the error occurred. This information is useful for analysis by the MySQL Cluster development team.

It is possible to configure the number of these trace files that will be created before old files are overwritten. `trace_id` is a number which is incremented for each successive trace file.

- `ndb_node_id_trace.log.next` is the file that keeps track of the next trace file number to be assigned.
- `ndb_node_id_out.log` is a file containing any data output by the **ndbd** process. This file is created only if **ndbd** is started as a daemon.
- `ndb_node_id.pid` is a file containing the process ID of the **ndbd** process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.
- `ndb_node_id_signal.log` is a file used only in debug versions of **ndbd**, where it is possible to trace all incoming, outgoing, and internal messages with their data in the **ndbd** process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the `.pid` file remains in effect even after the process has terminated.

To start **ndbd**, it may also be necessary to specify the hostname of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

See [Section 15.4.4.2, “The Cluster connectstring”](#), for additional information about this issue. [Section 15.6.5, “Command Options for MySQL Cluster Processes”](#), describes other options for **ndbd**.

When **ndbd** starts, it actually initiates two processes. The first of these is called the “angel process”; its only job is to discover when the execution process has been completed, and then to restart the **ndbd** process if it is configured to do so. Thus, if you attempt to kill **ndbd** via the Unix **kill** command, it is necessary to kill both processes, beginning with the angel process. The preferred method of terminating an **ndbd** process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent activities. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the **ndbd** process. In a system performing a large number of operations, including updates, the **ndbd** process can consume up to 2 CPUs if permitted to do so. For a machine with many CPUs it is recommended to use several **ndbd** processes which belong to different node groups.

15.6.3. **ndb_mgmd**, the Management Server Process

The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

It is not strictly necessary to specify a connectstring when starting the management server. However, if you are using more than one management server, a connectstring should be provided and each node in the cluster should specify its node ID explicitly.

See [Section 15.4.4.2, “The Cluster connectstring”](#), for information about using connectstrings. [Section 15.6.5, “Command Options for MySQL Cluster Processes”](#), describes other options for **ndb_mgmd**.

The following files are created or used by **ndb_mgmd** in its starting directory, and are placed in the `DataDir` as specified in the `config.ini` configuration file. In the list that follows, *node_id* is the unique node identifier.

- `config.ini` is the configuration file for the cluster as a whole. This file is created by the user and read by the management server. [Section 15.4, “MySQL Cluster Configuration”](#), discusses how to set up this file.
- `ndb_node_id_cluster.log` is the cluster events log file. Examples of such events include checkpoint startup and completion, node startup events, node failures, and levels of memory usage. A complete listing of cluster events with descriptions may be found in [Section 15.7, “Management of MySQL Cluster”](#).

When the size of the cluster log reaches one million bytes, the file is renamed to `ndb_node_id_cluster.log.seq_id`, where *seq_id* is the sequence number of the cluster log file. (For example: If files with the sequence numbers 1, 2, and 3 already exist, the next log file is named using the number 4.)

- `ndb_node_id_out.log` is the file used for `stdout` and `stderr` when running the management server as a daemon.
- `ndb_node_id.pid` is the process ID file used when running the management server as a daemon.

15.6.4. **ndb_mgm**, the Management Client Process

The management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client accesses the management server using a C API. Advanced users can also employ this API for programming dedicated management processes to perform tasks similar to those performed by **ndb_mgm**.

To start the management client, it is necessary to supply the hostname and port number of the management server:

```
shell> ndb_mgm [host_name [port_num]]
```

For example:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

The default hostname and port number are localhost and 1186, respectively.

Additional information about using **ndb_mgm** can be found in [Section 15.6.5.4, “Command Options for ndb_mgm”](#), and [Section 15.7.2, “Commands in the Management Client”](#).

15.6.5. Command Options for MySQL Cluster Processes

All MySQL Cluster executables (except for **mysqld**) take the options described in this section. Users of earlier MySQL Cluster versions should note that some of these options have been changed from those in MySQL 4.1 Cluster to make them consistent with one another as well as with **mysqld**. You can use the `--help` option to view a list of supported options.

The following sections describe options specific to individual NDB programs.

- `--help --usage, -?`

Prints a short list with descriptions of the available command options.

- `--connect-string=connect_string, -c connect_string`

connect_string sets the connectstring to the management server as a command option.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1
```

- `--debug[=options]`

This option can only be used for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the **mysqld** process.

- `--execute=command -e command`

Can be used to send a command to a Cluster executable from the system shell. For example, either of the following:

```
shell> ndb_mgm -e show
```

or

```
shell> ndb_mgm --execute="SHOW"
```

is equivalent to

```
NDB> SHOW;
```

This is analogous to how the `--execute` or `-e` option works with the **mysql** command-line client. See [Section 4.3.1, “Using Options on the Command Line”](#).

- `--version, -V`

Prints the version number of the **ndbd** process. The version number is the MySQL Cluster version number. The version number is relevant because not all versions can be used together, and the MySQL Cluster startup process verifies that the versions of the binaries being used can co-exist in the same cluster. This is also important when performing an online (rolling) software upgrade or downgrade of MySQL Cluster. (See [Section 15.5.1, “Performing a Rolling Restart of the Cluster”](#)).

15.6.5.1. MySQL Cluster-Related Command Options for **mysqld**

- `--ndb-connectstring=connect_string`

When using the `NDB Cluster` storage engine, this option specifies the management server that distributes cluster configuration data.

- `--ndbcluster`

The `NDB Cluster` storage engine is necessary for using MySQL Cluster. If a **mysqld** binary includes support for the `NDB Cluster` storage engine, the engine is disabled by default. Use the `--ndbcluster` option to enable it. Use `--skip-ndbcluster` to explicitly disable the engine.

15.6.5.2. Command Options for **ndbd**

For options common to NDB programs, see [Section 15.6.5, “Command Options for MySQL Cluster Processes”](#).

- `--daemon, -d`

Instructs **ndbd** to execute as a daemon process. This is the default behavior. `--nodaemon` can be used to not start the process as a daemon.

- `--initial`

Instructs **ndbd** to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of **ndbd**. It also re-creates recovery log files. Note that on some operating systems this process can take a substantial amount of time.

An `--initial` start is to be used only the very first time that the **ndbd** process is started because it removes all files from the Cluster filesystem and re-creates all REDO log files. The exceptions to this rule are:

- When performing a software upgrade which has changed the contents of any files.
- When restarting the node with a new version of **ndbd**.
- As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case, be aware that this node can no longer be used to restore data due to the destruction of the datafiles.

This option does not affect any backup files that have already been created by the affected node.

- `--initial-start`

This option is used when performing a partial initial start of the cluster. Each node should be started with this option, as well as `--no-wait-nodes`.

For example, suppose you have a 4-node cluster whose data nodes have the IDs 2, 3, 4, and 5, and you wish to perform a partial initial start using only nodes 2, 4, and 5 — that is, omitting node 3:

```
ndbd --ndbd-nodeid=2 --no-wait-nodes=3 --initial-start
```

```
ndbd --ndbd-nodeid=4 --no-wait-nodes=3 --initial-start
ndbd --ndbd-nodeid=5 --no-wait-nodes=3 --initial-start
```

This option was added in MySQL 5.0.21.

- `--nowait-nodes=node_id_1[, node_id_2[, ...]]`

This option takes a list of data nodes which for which the cluster will not wait for before starting.

This can be used to start the cluster in a partitioned state. For example, to start the cluster with only half of the data nodes (nodes 2, 3, 4, and 5) running in a 4-node cluster, you can start each **ndbd** process with `--nowait-nodes=3,5`. In this case, the cluster starts as soon as nodes 2 and 4 connect, and does *not* wait `StartPartitionedTimeout` milliseconds for nodes 3 and 5 to connect as it would otherwise.

If you wanted to start up the same cluster as in the previous example without one **ndbd** — say, for example, that the host machine for node 3 has suffered a hardware failure — then start nodes 2, 4, and 5 with `--no-wait-nodes=3`. Then the cluster will start as soon as nodes 2, 4, and 5 connect and will not wait for node 3 to start.

This option was added in MySQL 5.0.21.

- `--nodaemon`

Instructs **ndbd** not to start as a daemon process. This is useful when **ndbd** is being debugged and you want output to be redirected to the screen.

- `--nostart`

Instructs **ndbd** not to start automatically. When this option is used, **ndbd** connects to the management server, obtains configuration data from it, and initializes communication objects. However, it does not actually start the execution engine until specifically requested to do so by the management server. This can be accomplished by issuing the proper command to the management client.

15.6.5.3. Command Options for `ndb_mgmd`

For options common to NDB programs, see [Section 15.6.5, “Command Options for MySQL Cluster Processes”](#).

- `--config-file=filename, -f filename,`

Instructs the management server as to which file it should use for its configuration file. This option must be specified. The filename defaults to `config.ini`.

Note: This option also can be given as `-c file_name`, but this shortcut is obsolete and should *not* be used in new installations.

- `--daemon, -d`

Instructs **ndb_mgmd** to start as a daemon process. This is the default behavior.

- `--nodaemon`

Instructs **ndb_mgmd** not to start as a daemon process.

15.6.5.4. Command Options for **ndb_mgm**

For options common to NDB programs, see [Section 15.6.5, “Command Options for MySQL Cluster Processes”](#).

- `--try-reconnect=number`

If the connection to the management server is broken, the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to *number* before giving up and reporting an error instead.

15.7. Management of MySQL Cluster

Managing a MySQL Cluster involves a number of tasks, the first of which is to configure and start MySQL Cluster. This is covered in [Section 15.4, “MySQL Cluster Configuration”](#), and [Section 15.6, “Process Management in MySQL Cluster”](#).

The following sections cover the management of a running MySQL Cluster.

There are essentially two methods of actively managing a running MySQL Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped, and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_node_id_cluster.log` in the management server's `DataDir` directory. (Recall that `node_id` represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by **ndbd**. It is also possible to send cluster log entries to a Unix system log.

15.7.1. MySQL Cluster Startup Phases

This section describes the steps involved when the cluster is started.

There are several different startup types and modes, as shown here:

- **Initial Start:** The cluster starts with a clean filesystem on all data nodes. This occurs either when the cluster started for the very first time, or when it is restarted using the `--initial` option.
- **System Restart:** The cluster starts and reads data stored in the data nodes. This occurs when the cluster has been shut down after having been in use, when it is desired for the cluster to resume operations from the point where it left off.
- **Node Restart:** This is the online restart of a cluster node while the cluster itself is running.
- **Initial Node Restart:** This is the same as a node restart, except that the

node is reinitialized and started with a clean filesystem.

Prior to startup, each data node (`ndbd` process) must be initialized. Initialization consists of the following steps:

1. Obtain a Node ID.
2. Fetch configuration data.
3. Allocate ports to be used for inter-node communications.
4. Allocate memory according to settings obtained from the configuration file.

When a data node or SQL node first connects to the management node, it reserves a cluster node ID. To make sure that no other node allocates the same node ID, this ID is retained until the node has managed to connect to the cluster and at least one **ndbd** reports that this node is connected. This retention of the node ID is guarded by the connection between the node in question and **ndb_mgmd**.

Normally, in the event of a problem with the node, the node disconnects from the management server, the socket used for the connection is closed, and the reserved node ID is freed. However, if a node is disconnected abruptly — for example, due to a hardware failure in one of the cluster hosts, or because of network issues — the normal closing of the socket by the operating system may not take place. In this case, the node ID continues to be reserved and not released until a TCP timeout occurs 10 or so minutes later.

To take care of this problem, you can use `PURGE STALE SESSIONS`. Running this statement forces all reserved node IDs to be checked; any that are not being used by nodes actually connected to the cluster are then freed.

Beginning with MySQL 5.0.22, timeout handling of node ID assignments is implemented. This performs the ID usage checks automatically after approximately 20 seconds, so that `PURGE STALE SESSIONS` should no longer be necessary in a normal Cluster start.

After each data node has been initialized, the cluster startup process can proceed. The stages which the cluster goes through during this process are listed here:

- **Stage 0**

Clear the cluster filesystem. This stage occurs *only* if the cluster was started with the `--initial` option.

- **Stage 1**

This stage sets up Cluster connections, establishes inter-node communications are established, and starts Cluster heartbeats.

- **Stage 2**

The arbitrator node is elected. If this is a system restart, the cluster determines the latest restorable global checkpoint.

- **Stage 3**

This stage initializes a number of internal cluster variables.

- **Stage 4**

For an initial start or initial node restart, the redo log files are created. The number of these files is equal to `NoOfFragmentLogFiles`.

For a system restart:

- Read schema or schemas.
- Read data from the local checkpoint and undo logs.
- Apply all redo information until the latest restorable global checkpoint has been reached.

For a node restart, find the tail of the redo log.

- **Stage 5**

If this is an initial start, create the `SYSTAB_0` and `NDB$EVENTS` internal system tables.

For a node restart or an initial node restart:

1. The node is included in transaction handling operations.
2. The node schema is compared with that of the master and synchronized with it.
3. Synchronize data received in the form of `INSERT` from the other data nodes in this node's node group.
4. In all cases, wait for complete local checkpoint as determined by the arbitrator.

- **Stage 6**

Update internal variables.

- **Stage 7**

Update internal variables.

- **Stage 8**

In a system restart, rebuild all indexes.

- **Stage 9**

Update internal variables.

- **Stage 10**

At this point in a node restart or initial node restart, APIs may connect to the node and begin to receive events.

- **Stage 11**

At this point in a node restart or initial node restart, event delivery is handed over to the node joining the cluster. The newly-joined node takes over responsibility for delivering its primary data to subscribers.

After this process is completed for an initial start or system restart, transaction handling is enabled. For a node restart or initial node restart, completion of the startup process means that the node may now act as a transaction coordinator.

15.7.2. Commands in the Management Client

In addition to the central configuration file, a cluster may also be controlled through a command-line interface available through the management client **ndb_mgm**. This is the primary administrative interface to a running cluster.

Commands for the event logs are given in [Section 15.7.3, “Event Reports Generated in MySQL Cluster”](#); commands for creating backups and restoring from backup are provided in [Section 15.8, “On-line Backup of MySQL Cluster”](#).

The management client has the following basic commands. In the listing that follows, *node_id* denotes either a database node ID or the keyword ALL, which indicates that the command should be applied to all of the cluster's data nodes.

- HELP

Displays information on all available commands.

- SHOW

Displays information on the cluster's status.

Note: In a cluster where multiple management nodes are in use, this command displays information only for data nodes that are actually connected to the current management server.

- *node_id* START

Starts the data node identified by *node_id* (or all data nodes).

Beginning with MySQL 5.0.19, this command can also be used to start individual management nodes. **Note:** ALL START continues to affect data nodes only.

- *node_id* STOP

Stops the data node identified by *node_id* (or all data nodes).

Beginning with MySQL 5.0.19, this command can also be used to stop individual management nodes. **Note:** ALL STOP continues to affect data

nodes only.

- `node_id RESTART [-N] [-I]`

Restarts the data node identified by *node_id* (or all data nodes).

- `node_id STATUS`

Displays status information for the data node identified by *node_id* (or for all data nodes).

- `ENTER SINGLE USER MODE node_id`

Enters single-user mode, whereby only the MySQL server identified by the node ID *node_id* is allowed to access the database.

- `EXIT SINGLE USER MODE`

Exits single-user mode, allowing all SQL nodes (that is, all running **mysqld** processes) to access the database.

- `QUIT`

Terminates the management client.

- `SHUTDOWN`

Shuts down all cluster nodes, except for SQL nodes, and exits.

15.7.3. Event Reports Generated in MySQL Cluster

In this section, we discuss the types of event logs provided by MySQL Cluster, and the types of events that are logged.

MySQL Cluster provides two types of event log. These are the *cluster log*, which includes events generated by all cluster nodes, and the *node logs*, which are local to each data node.

Output generated by cluster event logging can have multiple destinations including a file, the management server console window, or `syslog`. Output

generated by node event logging is written to the data node's console window.

Both types of event logs can be set to log different subsets of events.

Note: The cluster log is the log recommended for most uses because it provides logging information for an entire cluster in a single file. Node logs are intended to be used only during application development, or for debugging application code.

Each reportable event can be distinguished according to three different criteria:

- *Category:* This can be any one of the following values: STARTUP, SHUTDOWN, STATISTICS, CHECKPOINT, NODERESTART, CONNECTION, ERROR, or INFO.
- *Priority:* This is represented by one of the numbers from 1 to 15 inclusive, where 1 indicates “most important” and 15 “least important.”
- *Severity Level:* This can be any one of the following values: ALERT, CRITICAL, ERROR, WARNING, INFO, or DEBUG.

Both the cluster log and the node log can be filtered on these properties.

15.7.3.1. Logging Management Commands

The following management commands are related to the cluster log:

- CLUSTERLOG ON
Turns the cluster log on.
- CLUSTERLOG OFF
Turns the cluster log off.
- CLUSTERLOG INFO
Provides information about cluster log settings.
- node_id CLUSTERLOG *category=threshold*

Logs *category* events with priority less than or equal to *threshold* in the cluster log.

- CLUSTERLOG FILTER *severity_level*

Toggles cluster logging of events of the specified *severity_level*.

The following table describes the default setting (for all data nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, it is reported in the cluster log.

Note that events are reported per data node, and that the threshold can be set to different values on different nodes.

Category	Default threshold (All data nodes)
STARTUP	7
SHUTDOWN	7
STATISTICS	7
CHECKPOINT	7
NODERESTART	7
CONNECTION	7
ERROR	15
INFO	7

Thresholds are used to filter events within each category. For example, a STARTUP event with a priority of 3 is not logged unless the threshold for STARTUP is changed to 3 or lower. Only events with priority 3 or lower are sent if the threshold is 3.

The following table shows the event severity levels. (**Note:** These correspond to Unix syslog levels, except for LOG_EMERG and LOG_NOTICE, which are not used or mapped.)

1	ALERT	A condition that should be corrected immediately, such as a corrupted system database
2	CRITICAL	Critical conditions, such as device errors or insufficient resources

3	ERROR	Conditions that should be corrected, such as configuration errors
4	WARNING	Conditions that are not errors, but that might require special handling
5	INFO	Informational messages
6	DEBUG	Debugging messages used for NDB Cluster development

Event severity levels can be turned on or off (using `CLUSTERLOG FILTER` — see above). If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

15.7.3.2. Log Events

An event report reported in the event logs has the following format:

```
datetime [string] severity -- message
```

For example:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

This section discusses all reportable events, ordered by category and severity level within each category.

In the event descriptions, GCP and LCP mean “Global Checkpoint” and “Local Checkpoint,” respectively.

CONNECTION Events

These events are associated with connections between Cluster nodes.

Event	Priority	Severity Level	Description
data nodes connected	8	INFO	Data nodes connected
data nodes disconnected	8	INFO	Data nodes disconnected
Communication			SQL node or data node connection

closed	8	INFO	closed
Communication opened	8	INFO	SQL node or data node connection opened

CHECKPOINT Events

The logging messages shown here are associated with checkpoints.

Event	Priority	Severity Level	Description
LCP stopped in calc keep GCI	0	ALERT	LCP stopped
Local checkpoint fragment completed	11	INFO	LCP on a fragment has been completed
Global checkpoint completed	10	INFO	GCP finished
Global checkpoint started	9	INFO	Start of GCP: REDO log is written to disk
Local checkpoint completed	8	INFO	LCP completed normally
Local checkpoint started	7	INFO	Start of LCP: data written to disk
Report undo log blocked	7	INFO	UNDO logging blocked; buffer near overflow

STARTUP Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

Event	Priority	Severity Level	Description
Internal start signal received STTORY	15	INFO	Blocks received after completion of restart

Undo records executed	15	INFO	
New REDO log started	10	INFO	GCI keep x , newest restorable GCI y
New log started	10	INFO	Log part x , start MB y , stop MB z
Node has been refused for inclusion in the cluster	8	INFO	Node cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem
data node neighbors	8	INFO	Shows neighboring data nodes
data node start phase x completed	4	INFO	A data node start phase has been completed
Node has been successfully included into the cluster	3	INFO	Displays the node, managing node, and dynamic ID
data node start phases initiated	1	INFO	NDB Cluster nodes starting
data node all start phases completed	1	INFO	NDB Cluster nodes started
data node shutdown initiated	1	INFO	Shutdown of data node has commenced
data node shutdown aborted	1	INFO	Unable to shut down data node normally

NODERESTART Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

Event	Priority	Severity Level	Description
Node failure phase completed	8	ALERT	Reports completion of node failure phases
Node has failed, node		ALERT	

state was x	8		Reports that a node has failed
Report arbitrator results	2	ALERT	<p>There are eight different possible results for arbitration attempts:</p> <ul style="list-style-type: none"> • Arbitration check failed — less than 1/2 nodes left • Arbitration check succeeded — node group majority • Arbitration check failed — missing node group • Network partitioning — arbitration required • Arbitration succeeded — affirmative response from node x • Arbitration failed - negative response from node x • Network partitioning - no arbitrator available • Network partitioning - no arbitrator configured
Completed copying a fragment	10	INFO	
Completed copying of dictionary information	8	INFO	
Completed copying distribution information	8	INFO	
Starting to copy fragments	8	INFO	

Completed copying all fragments	8	INFO	
GCP takeover started	7	INFO	
GCP takeover completed	7	INFO	
LCP takeover started	7	INFO	
LCP takeover completed (state = x)	7	INFO	
Report whether an arbitrator is found or not	6	INFO	<p>There are seven different possible outcomes when seeking an arbitrator:</p> <ul style="list-style-type: none"> • Management server restarts arbitration thread [state=x] • Prepare arbitrator node x [ticket=γ] • Receive arbitrator node x [ticket=γ] • Started arbitrator node x [ticket=γ] • Lost arbitrator node x - process failure [state=γ] • Lost arbitrator node x - process exit [state=γ] • Lost arbitrator node x <error msg> [state=γ]

STATISTICS Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

Event	Priority	Severity Level	Description
Report job scheduling statistics	9	INFO	Mean internal job scheduling statistics
Sent number of bytes	9	INFO	Mean number of bytes sent to node x
Received # of bytes	9	INFO	Mean number of bytes received from node x
Report transaction statistics	8	INFO	Numbers of: transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts
Report operations	8	INFO	Number of operations
Report table create	7	INFO	
Memory usage	5	INFO	Data and index memory usage (80%, 90%, and 100%)

ERROR Events

These events relate to Cluster errors and warnings. The presence of one or more of these generally indicates that a major malfunction or failure has occurred.

Event	Priority	Severity	Description
Dead due to missed heartbeat	8	ALERT	Node x declared “dead” due to missed heartbeat
Transporter errors	2	ERROR	
Transporter warnings	8	WARNING	
Missed heartbeats	8	WARNING	Node x missed heartbeat # Y
General warning events	2	WARNING	

INFO Events

These events provide general information about the state of the cluster and

activities associated with Cluster maintenance, such as logging and heartbeat transmission.

Event	Priority	Severity	Description
Sent heartbeat	12	INFO	Heartbeat sent to node x
Create log bytes	11	INFO	Log part, log file, MB
General information events	2	INFO	

15.7.3.3. Using CLUSTERLOG STATISTICS

The NDB management client's CLUSTERLOG STATISTICS command can provide a number of useful statistics in its output. The following statistics are reported by the transaction coordinator:

Statistic	Description (<i>Number of...</i>)
Trans. Count	Transactions attempted with this node as coordinator
Commit Count	Transactions committed with this node as coordinator
Read Count	Primary key reads (all)
Simple Read Count	Primary key reads reading the latest committed value
Write Count	Primary key writes (includes all INSERT, UPDATE, and DELETE operations)
AttrInfoCount	Data words used to describe all reads and writes received
Concurrent Operations	All concurrent operations ongoing at the moment the report is taken
Abort Count	Transactions with this node as coordinator that were aborted
Scans	Scans (all)
Range Scans	Index scans

The **ndbd** process has a scheduler that runs in an infinite loop. During each loop scheduler performs the following tasks:

1. Read any incoming messages from sockets into a job buffer.

2. Check whether there are any timed messages to be executed; if so, put these into the job buffer as well.
3. Execute (in a loop) any messages in the job buffer.
4. Send any distributed messages that were generated by executing the messages in the job buffer.
5. Wait for any new incoming messages.

The number of loops executed in the third step is reported as the `Mean Loop Counter`. This statistic increases in size as the utilisation of the TCP/IP buffer improves. You can use this to monitor performance as you add new processes to the cluster.

The `Mean send size` and `Mean receive size` statistics allow you to gauge the efficiency of writes and reads (respectively) between nodes. These values are given in bytes. Higher values mean a lower cost per byte sent or received; the maximum is 64k.

To generate a report of all cluster log statistics, you can use the following command in the NDB management client:

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

15.7.4. Single-User Mode

Single-user mode allows the database administrator to restrict access to the database system to a single MySQL server (SQL node). When entering single-user mode, all connections to all other MySQL servers are closed gracefully and all running transactions are aborted. No new transactions are allowed to be started.

Once the cluster has entered single-user mode, only the designated SQL node is granted access to the database.

You can use the **ALL STATUS** command to see when the cluster has entered single-user mode.

Example:

NDB> ENTER SINGLE USER MODE 5

After this command has executed and the cluster has entered single-user mode, the SQL node whose node ID is 5 becomes the cluster's only permitted user.

The node specified in the preceding command must be a MySQL Server node; An attempt to specify any other type of node will be rejected.

Note: When the preceding command is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command **EXIT SINGLE USER MODE** changes the state of the cluster's data nodes from single-user mode to normal mode. MySQL Servers waiting for a connection (that is, for the cluster to become ready and available), are again permitted to connect. The MySQL Server denoted as the single-user SQL node continues to run (if still connected) during and after the state change.

Example:

NDB> EXIT SINGLE USER MODE

There are two recommended ways to handle a node failure when running in single-user mode:

- Method 1:
 1. Finish all single-user mode transactions
 2. Issue the **EXIT SINGLE USER MODE** command
 3. Restart the cluster's data nodes
- Method 2:

Restart database nodes prior to entering single-user mode.

15.8. On-line Backup of MySQL Cluster

This section describes how to create a backup and how to restore the database from a backup at a later time.

15.8.1. Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata:** the names and definitions of all database tables
- **Table records:** the data actually stored in the database tables at the time that the backup was made
- **Transaction log:** a sequential record telling how and when data was stored in the database

Each of these parts is saved on all nodes participating in the backup. During backup, each node saves these three parts into three files on disk:

- BACKUP-backup_id.*node_id*.ctl

A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- BACKUP-backup_id-0.*node_id*.data

A data file containing the table records, which are saved on a per-fragment basis. That is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states the tables to which the records belong. Following the list of records there is a footer containing a checksum for all records.

- BACKUP-backup_id.*node_id*.log

A log file containing records of committed transactions. Only transactions

on tables stored in the backup are stored in the log. Nodes involved in the backup save different records because different nodes host different database fragments.

In the listing above, *backup_id* stands for the backup identifier and *node_id* is the unique identifier for the node creating the file.

15.8.2. Using The Management Client to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See [Section 15.8.4, “Configuration for Cluster Backup”](#).)

Creating a backup using the management client involves the following steps:

1. Start the management client (**ndb_mgm**).
2. Execute the command `START BACKUP`.
3. The management client will reply with the message `Start of backup ordered`. This means that the management client has submitted the request to the cluster, but has not yet received any response.
4. The management client will reply `Backup backup_id started`, where *backup_id* is the unique identifier for this particular backup. (This identifier will also be saved in the cluster log, if it has not been configured otherwise.) This means that the cluster has received and processed the backup request. It does *not* mean that the backup has finished.
5. The management client will signal that the backup is finished with the message `Backup backup_id completed`.

To abort a backup that is already in progress:

1. Start the management client.
2. Execute the command `ABORT BACKUP backup_id`. The number *backup_id* is the identifier of the backup that was included in the response of the management client when the backup was started (in the message `Backup backup_id started`).

3. The management client will acknowledge the abort request with `Abort` of `backup backup_id` ordered; note that it has received no actual response to this request yet.
4. After the backup has been aborted, the management client will report `Backup backup_id` has been aborted for reason `XYZ`. This means that the cluster has terminated the backup and that all files related to this backup have been removed from the cluster filesystem.

It is also possible to abort a backup in progress from the system shell using this command:

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

Note: If there is no backup with ID `backup_id` running when it is aborted, the management client makes no explicit response. However, the fact that an invalid abort command was sent is indicated in the cluster log.

15.8.3. How to Restore a Cluster Backup

The cluster restoration program is implemented as a separate command-line utility **ndb_restore**, which reads the files created by the backup and inserts the stored information into the database. The restore program must be executed once for each set of backup files. That is, as many times as there were database nodes running when the backup was created.

The first time you run the **ndb_restore** restoration program, you also need to restore the metadata. In other words, you must re-create the database tables. (Note that the cluster should have an empty database when starting to restore a backup.) The restore program acts as an API to the cluster and therefore requires a free connection to connect to the cluster. This can be verified with the **ndb_mgm** command **SHOW** (you can accomplish this from a system shell using **ndb_mgm -e SHOW**). The `-c connectstring` option may be used to locate the MGM node (see [Section 15.4.4.2, “The Cluster connectstring”](#), for information on connectstrings). The backup files must be present in the directory given as an argument to the restoration program.

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID 12, created in a cluster with two database nodes having the node IDs 2 and 3, is to be

restored to a cluster with four nodes. Then **ndb_restore** must be run twice — once for each database node in the cluster where the backup was taken. However, **ndb_restore** cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version. See [Section 15.5.2, “Cluster Upgrade and Downgrade Compatibility”](#), for more information.

Note: For rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. However, the data files must always be applied before the logs.

15.8.4. Configuration for Cluster Backup

Four configuration parameters are essential for backup:

- BackupDataBufferSize

The amount of memory used to buffer data before it is written to disk.

- BackupLogBufferSize

The amount of memory used to buffer log records before these are written to disk.

- BackupMemory

The total memory allocated in a database node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.

- BackupWriteSize

The size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

More detailed information about these parameters can be found in [Section 15.4.4.5, “Defining Data Nodes”](#).

15.8.5. Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or disk space. You should check that there is enough memory allocated for the backup. **Important:** If you have set `BackupDataBufferSize` and `BackupLogBufferSize` and their sum is greater than 4MB, then you must also set `BackupMemory` as well. See [BackupMemory](#).

You should also make sure that there is sufficient space on the hard drive partition of the backup target.

NDB does not support repeatable reads, which can cause problems with the restoration process. Although the backup process is “hot”, restoring a MySQL Cluster from backup is not a 100% “hot” process. This is due to the fact that, for the duration of the restore process, running transactions get non-repeatable reads from the restored data. This means that the state of the data is inconsistent while the restore is in progress.

15.9. Using High-Speed Interconnects with MySQL Cluster

Even before design of NDB Cluster began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, NDB Cluster was designed from the very beginning to allow for the use of a number of different data transport mechanisms. In this Manual, we use the term *transporter* for these.

The MySQL Cluster codebase includes support for four different transporters:

- *TCP/IP using 100 Mbps or gigabit Ethernet*, as discussed in [Section 15.4.4.7, “Cluster TCP/IP Connections”](#).
- *Direct (machine-to-machine) TCP/IP*; although this transporter uses the same TCP/IP protocol as mentioned in the previous item, it requires setting up the hardware differently and is configured differently as well. For this reason, it is considered a separate transport mechanism for MySQL Cluster. See [Section 15.4.4.8, “TCP/IP Connections Using Direct Connections”](#), for details.
- *Shared memory (SHM)*. For more information about SHM, see [Section 15.4.4.9, “Shared-Memory Connections”](#).
- *Scalable Coherent Interface (SCI)*, as described in the next section of this chapter, [Section 15.4.4.10, “SCI Transport Connections”](#).

Most users today employ TCP/IP over Ethernet because it is ubiquitous. TCP/IP is also by far the best-tested transporter for use with MySQL Cluster.

We are working to make sure that communication with the **ndbd** process is made in “chunks” that are as large as possible because this benefits all types of data transmission.

For users who desire it, it is also possible to use cluster interconnects to enhance performance even further. There are two ways to achieve this: Either a custom transporter can be designed to handle this case, or you can use socket

implementations that bypass the TCP/IP stack to one extent or another. We have experimented with both of these techniques using the SCI (Scalable Coherent Interface) technology developed by [Dolphin](#).

15.9.1. Configuring MySQL Cluster to use SCI Sockets

In this section, we show how to adapt a cluster configured for normal TCP/IP communication to use SCI Sockets instead. This documentation is based on SCI Sockets version 2.3.0 as of 01 October 2004.

Prerequisites

Any machines with which you wish to use SCI Sockets must be equipped with SCI cards.

It is possible to use SCI Sockets with any version of MySQL Cluster. No special builds are needed because it uses normal socket calls which are already available in MySQL Cluster. However, SCI Sockets are currently supported only on the Linux 2.4 and 2.6 kernels. SCI Transporters have been tested successfully on additional operating systems although we have verified these only with Linux 2.4 to date.

There are essentially four requirements for SCI Sockets:

- Building the SCI Socket libraries.
- Installation of the SCI Socket kernel libraries.
- Installation of one or two configuration files.
- The SCI Socket kernel library must be enabled either for the entire machine or for the shell where the MySQL Cluster processes are started.

This process needs to be repeated for each machine in the cluster where you plan to use SCI Sockets for inter-node communication.

Two packages need to be retrieved to get SCI Sockets working:

- The source code package containing the DIS support libraries for the SCI Sockets libraries.

- The source code package for the SCI Socket libraries themselves.

Currently, these are available only in source code format. The latest versions of these packages at the time of this writing were available as (respectively)

`DIS_GPL_2_5_0_SEP_10_2004.tar.gz` and

`SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz`. You should be able to find these (or possibly newer versions) at <http://www.dolphinics.no/support/downloads.html>.

Package Installation

Once you have obtained the library packages, the next step is to unpack them into appropriate directories, with the SCI Sockets library unpacked into a directory below the DIS code. Next, you need to build the libraries. This example shows the commands used on Linux/x86 to perform this task:

```
shell> tar xzf DIS_GPL_2_5_0_SEP_10_2004.tar.gz
shell> cd DIS_GPL_2_5_0_SEP_10_2004/src/
shell> tar xzf ../../SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz
shell> cd ../adm/bin/Linux_pkgs
shell> ./make_PSB_66_release
```

It is possible to build these libraries for some 64-bit procesors. To build the libraries for Opteron CPUs using the 64-bit extensions, run

`make_PSB_66_X86_64_release` rather than **`make_PSB_66_release`**. If the build is made on an Itanium machine, you should use **`make_PSB_66_IA64_release`**. The X86-64 variant should work for Intel EM64T architectures but this has not yet (to our knowledge) been tested.

Once the build process is complete, the compiled libraries will be found in a zipped tar file with a name along the lines of `DIS-<operating-system>-time-date`. It is now time to install the package in the proper place. In this example we will place the installation in `/opt/DIS`. (**Note:** You will most likely need to run the following as the system root user.)

```
shell> cp DIS_Linux_2.4.20-8_181004.tar.gz /opt/
shell> cd /opt
shell> tar xzf DIS_Linux_2.4.20-8_181004.tar.gz
shell> mv DIS_Linux_2.4.20-8_181004 DIS
```

Network Configuration

Now that all the libraries and binaries are in their proper place, we need to

ensure that the SCI cards have proper node IDs within the SCI address space.

It is also necessary to decide on the network structure before proceeding. There are three types of network structures which can be used in this context:

- A simple one-dimensional ring
- One or more SCI switches with one ring per switch port
- A two- or three-dimensional torus.

Each of these topologies has its own method for providing node IDs. We discuss each of them in brief.

A simple ring uses node IDs which are non-zero multiples of 4: 4, 8, 12,...

The next possibility uses SCI switches. An SCI switch has 8 ports, each of which can support a ring. It is necessary to make sure that different rings use different node ID spaces. In a typical configuration, the first port uses node IDs below 64 (4 – 60), the next 64 node IDs (68 – 124) are assigned to the next port, and so on, with node IDs 452 – 508 being assigned to the eighth port.

Two- and three-dimensional torus network structures take into account where each node is located in each dimension, incrementing by 4 for each node in the first dimension, by 64 in the second dimension, and (where applicable) by 1024 in the third dimension. See [Dolphin's Web site](#) for more thorough documentation.

In our testing we have used switches, although most large cluster installations use 2- or 3-dimensional torus structures. The advantage provided by switches is that, with dual SCI cards and dual switches, it is possible to build with relative ease a redundant network where the average failover time on the SCI network is on the order of 100 microseconds. This is supported by the SCI transporter in MySQL Cluster and is also under development for the SCI Socket implementation.

Failover for the 2D/3D torus is also possible but requires sending out new routing indexes to all nodes. However, this requires only 100 milliseconds or so to complete and should be acceptable for most high-availability cases.

By placing cluster data nodes properly within the switched architecture, it is possible to use 2 switches to build a structure whereby 16 computers can be interconnected and no single failure can hinder more than one of them. With 32 computers and 2 switches it is possible to configure the cluster in such a manner that no single failure can cause the loss of more than two nodes; in this case, it is also possible to know which pair of nodes is affected. Thus, by placing the two nodes in separate node groups, it is possible to build a “safe” MySQL Cluster installation.

To set the node ID for an SCI card use the following command in the `/opt/DIS/sbin` directory. In this example, `-c 1` refers to the number of the SCI card (this is always 1 if there is only 1 card in the machine); `-a 0` refers to adapter 0; and 68 is the node ID:

```
shell> ./sciconfig -c 1 -a 0 -n 68
```

If you have multiple SCI cards in the same machine, you can determine which card has which slot by issuing the following command (again we assume that the current working directory is `/opt/DIS/sbin`):

```
shell> ./sciconfig -c 1 -gsn
```

This will give you the SCI card's serial number. Then repeat this procedure with `-c 2`, and so on, for each card in the machine. Once you have matched each card with a slot, you can set node IDs for all cards.

After the necessary libraries and binaries are installed, and the SCI node IDs are set, the next step is to set up the mapping from hostnames (or IP addresses) to SCI node IDs. This is done in the SCI sockets configuration file, which should be saved as `/etc/sci/scisock.conf`. In this file, each SCI node ID is mapped through the proper SCI card to the hostname or IP address that it is to communicate with. Here is a very simple example of such a configuration file:

```
#host          #nodeId
alpha          8
beta           12
192.168.10.20  16
```

It is also possible to limit the configuration so that it applies only to a subset of the available ports for these hosts. An additional configuration file `/etc/sci/scisock_opt.conf` can be used to accomplish this, as shown here:

#-key	-type	-values
EnablePortsByDefault	yes	
EnablePort	tcp	2200
DisablePort	tcp	2201
EnablePortRange	tcp	2202 2219
DisablePortRange	tcp	2220 2231

Driver Installation

With the configuration files in place, the drivers can be installed.

First, the low-level drivers and then the SCI socket driver need to be installed:

```
shell> cd DIS/sbin/
shell> ./drv-install add PSB66
shell> ./scisocket-install add
```

If desired, the installation can be checked by invoking a script which verifies that all nodes in the SCI socket configuration files are accessible:

```
shell> cd /opt/DIS/sbin/
shell> ./status.sh
```

If you discover an error and need to change the SCI socket configuration, it is necessary to use **ksocketconfig** to accomplish this task:

```
shell> cd /opt/DIS/util
shell> ./ksocketconfig -f
```

Testing the Setup

To ensure that SCI sockets are actually being used, you can employ the **latency_bench** test program. Using this utility's server component, clients can connect to the server to test the latency of the connection. Determining whether SCI is enabled should be fairly simple from observing the latency. (**Note:** Before using **latency_bench**, it is necessary to set the LD_PRELOAD environment variable as shown later in this section.)

To set up a server, use the following:

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -server
```

To run a client, use **latency_bench** again, except this time with the **-client**

option:

```
shell> cd /opt/DIS/bin/socket  
shell> ./latency_bench -client server_hostname
```

SCI socket configuration should now be complete and MySQL Cluster ready to use both SCI Sockets and the SCI transporter (see [Section 15.4.4.10, “SCI Transport Connections”](#)).

Starting the Cluster

The next step in the process is to start MySQL Cluster. To enable usage of SCI Sockets it is necessary to set the environment variable `LD_PRELOAD` before starting `ndbd`, `mysqld`, and `ndb_mgmd`. This variable should point to the kernel library for SCI Sockets.

To start `ndbd` in a bash shell, do the following:

```
bash-shell> export LD_PRELOAD=/opt/DIS/lib/libkscisock.so  
bash-shell> ndbd
```

In a tcsh environment the same thing can be accomplished with:

```
tcsh-shell> setenv LD_PRELOAD=/opt/DIS/lib/libkscisock.so  
tcsh-shell> ndbd
```

Note: MySQL Cluster can use only the kernel variant of SCI Sockets.

15.9.2. Understanding the Impact of Cluster Interconnects

The `ndbd` process has a number of simple constructs which are used to access the data in a MySQL Cluster. We have created a very simple benchmark to check the performance of each of these and the effects which various interconnects have on their performance.

There are four access methods:

- **Primary key access**

This is access of a record through its primary key. In the simplest case, only one record is accessed at a time, which means that the full cost of setting up

a number of TCP/IP messages and a number of costs for context switching are borne by this single request. In the case where multiple primary key accesses are sent in one batch, those accesses share the cost of setting up the necessary TCP/IP messages and context switches. If the TCP/IP messages are for different destinations, additional TCP/IP messages need to be set up.

- **Unique key access**

Unique key accesses are similar to primary key accesses, except that a unique key access is executed as a read on an index table followed by a primary key access on the table. However, only one request is sent from the MySQL Server, and the read of the index table is handled by **ndbd**. Such requests also benefit from batching.

- **Full table scan**

When no indexes exist for a lookup on a table, a full table scan is performed. This is sent as a single request to the **ndbd** process, which then divides the table scan into a set of parallel scans on all cluster **ndbd** processes. In future versions of MySQL Cluster, an SQL node will be able to filter some of these scans.

- **Range scan using ordered index**

When an ordered index is used, it performs a scan in the same manner as the full table scan, except that it scans only those records which are in the range used by the query transmitted by the MySQL server (SQL node). All partitions are scanned in parallel when all bound index attributes include all attributes in the partitioning key.

To check the base performance of these access methods, we have developed a set of benchmarks. One such benchmark, **testReadPerf**, tests simple and batched primary and unique key accesses. This benchmark also measures the setup cost of range scans by issuing scans returning a single record. There is also a variant of this benchmark which uses a range scan to fetch a batch of records.

In this way, we can determine the cost of both a single key access and a single record scan access, as well as measure the impact of the communication media used, on base access methods.

In our tests, we ran the base benchmarks for both a normal transporter using TCP/IP sockets and a similar setup using SCI sockets. The figures reported in the following table are for small accesses of 20 records per access. The difference between serial and batched access decreases by a factor of 3 to 4 when using 2KB records instead. SCI Sockets were not tested with 2KB records. Tests were performed on a cluster with 2 data nodes running on 2 dual-CPU machines equipped with AMD MP1900+ processors.

Access Type	TCP/IP Sockets	SCI Socket
Serial pk access	400 microseconds	160 microseconds
Batched pk access	28 microseconds	22 microseconds
Serial uk access	500 microseconds	250 microseconds
Batched uk access	70 microseconds	36 microseconds
Indexed eq-bound	1250 microseconds	750 microseconds
Index range	24 microseconds	12 microseconds

We also performed another set of tests to check the performance of SCI Sockets *vis-à-vis* that of the SCI transporter, and both of these as compared with the TCP/IP transporter. All these tests used primary key accesses either serially and multi-threaded, or multi-threaded and batched.

The tests showed that SCI sockets were about 100% faster than TCP/IP. The SCI transporter was faster in most cases compared to SCI sockets. One notable case occurred with many threads in the test program, which showed that the SCI transporter did not perform very well when used for the **mysqld** process.

Our overall conclusion was that, for most benchmarks, using SCI sockets improves performance by approximately 100% over TCP/IP, except in rare instances when communication performance is not an issue. This can occur when scan filters make up most of processing time or when very large batches of primary key accesses are achieved. In that case, the CPU processing in the **ndbd** processes becomes a fairly large part of the overhead.

Using the SCI transporter instead of SCI Sockets is only of interest in communicating between **ndbd** processes. Using the SCI transporter is also only of interest if a CPU can be dedicated to the **ndbd** process because the SCI transporter ensures that this process will never go to sleep. It is also important to

ensure that the **ndbd** process priority is set in such a way that the process does not lose priority due to running for an extended period of time, as can be done by locking processes to CPUs in Linux 2.6. If such a configuration is possible, the **ndbd** process will benefit by 10–70% as compared with using SCI sockets. (The larger figures will be seen when performing updates and probably on parallel scan operations as well.)

There are several other optimized socket implementations for computer clusters, including Myrinet, Gigabit Ethernet, Infiniband and the VIA interface. We have tested MySQL Cluster so far only with SCI sockets. See [Section 15.9.1, “Configuring MySQL Cluster to use SCI Sockets”](#) for information on how to set up SCI sockets using ordinary TCP/IP for MySQL Cluster.

15.10. Known Limitations of MySQL Cluster

In this section, we provide a list of known limitations in MySQL Cluster releases in the 5.0.x series compared to features available when using the MyISAM and InnoDB storage engines. Currently, there are no plans to address these in coming releases of MySQL 5.0; however, we will attempt to supply fixes for these issues in subsequent release series. If you check the “Cluster” category in the MySQL bugs database at <http://bugs.mysql.com>, you can find known bugs which (if marked “5.0”) we intend to correct in upcoming releases of MySQL 5.0.

The list here is intended to be complete with respect to the conditions just set forth. You can report any discrepancies that you encounter to the MySQL bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#). If we do not plan to fix the problem in MySQL 5.0, we will add it to the list.

(**Note:** See the end of this section for a list of issues in MySQL 4.1 Cluster that have been resolved in the current version.)

- **Noncompliance in syntax** (resulting in errors when running existing applications):
 - Text indexes are not supported. That is, you cannot create indexes on columns of any of the TEXT datatypes, nor does the NDB storage engine support FULLTEXT indexes (these are supported by MyISAM only). However, you can index CHAR or VARCHAR columns of NDB tables.
 - A BIT column cannot be a primary key or part of a composite primary key.
 - Geometry datatypes (WKT and WKB) are not supported by the NDB storage engine prior to MySQL 5.0.16. (Note that spatial indexes are still not supported in MySQL 5.0.16 and newer.)
 - In MySQL 5.0.19 and earlier, INSERT IGNORE, UPDATE IGNORE, and REPLACE are supported only for primary keys, but not for unique keys. One possible workaround is to remove the constraint by dropping the unique index, perform any inserts, and then add the unique index

again.

This limitation is removed for `INSERT IGNORE` and `REPLACE` in MySQL 5.0.20. (Bug #17431)

- **Non-compliance in limits or behavior** (may result in errors when running existing applications):

- **Error Reporting:**

- A duplicate key error returns the error message `ERROR 23000: Can't write; duplicate key in table 'tbl_name'`.
- Like other MySQL storage engines, the NDB storage engine can handle a maximum of one `AUTO_INCREMENT` column per table. However, in the case of a Cluster table with no explicit primary key, an `AUTO_INCREMENT` column is automatically defined and used as a “hidden” primary key. For this reason, you cannot define a table that has an explicit `AUTO_INCREMENT` column unless that column is also declared using the `PRIMARY KEY` option.

Attempting to create a table with an `AUTO_INCREMENT` column that is not the table's primary key, and using the NDB storage engine, fails with an error.

- **Transaction Handling:**

- NDB Cluster supports only the `READ COMMITTED` transaction isolation level.
- There is no partial rollback of transactions. A duplicate key or similar error results in a rollback of the entire transaction.
- **Important:** If a `SELECT` from a Cluster table includes a `BLOB`, `TEXT`, or `VARCHAR` column, the `READ COMMITTED` transaction isolation level is converted to a read with read lock. This is done to guarantee consistency, due to the fact that parts of the values stored in columns of these types are actually read from a separate table.

- As noted elsewhere in this chapter, MySQL Cluster does not handle large transactions well; it is better to perform a number of small transactions with a few operations each than to attempt a single large transaction containing a great many operations.

Among other considerations, large transactions require very large amounts of memory. Because of this, the transactional behaviour of a number of MySQL statements is effected as described in the following list:

- `TRUNCATE` is not transactional when used on NDB tables. If a `TRUNCATE` fails to empty the table, then it must be re-run until it is successful.
- `DELETE FROM` (even with no `WHERE` clause) is transactional. For tables containing a great many rows, you may find that performance is improved by using several `DELETE FROM . . . LIMIT . . .` statements to “chunk” the delete operation. If the objective is to empty the table, then you may wish to use `TRUNCATE` instead.
- `LOAD DATA INFILE` is not transactional. During such an operation the NDB engine can and does commit at will.

`LOAD DATA FROM MASTER` is not supported in MySQL Cluster.
- When copying a table as part of an `ALTER TABLE`, the creation of the copy is non-transactional. (In any case, this operation is rolled back when the copy is deleted.)
- **Node Start, Stop, or Restart:** Starting, stopping, or restarting a node may give rise to temporary errors causing some transactions to fail. These include the following cases:
 - When first starting a node, it is possible that you may see Error 1204 Temporary failure, distribution changed and similar temporary errors.
 - The stopping or failure of any data node can result in a

number of different node failure errors. (However, there should be no aborted transactions when performing a planned shutdown of the cluster.)

In either of these cases, any errors that are generated must be handled within the application. This should be done by retrying the transaction.

- A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in [Section 15.4.4, “Configuration File”](#). Most configuration parameters can be upgraded online. These hard limits include:

- Database memory size and index memory size (DataMemory and IndexMemory, respectively).

DataMemory is allocated as 32KB pages. As each DataMemory page is used, it is assigned to a specific table; once allocated, this memory cannot be freed except by dropping the table.

See [Section 15.4.4.5, “Defining Data Nodes”](#), for further information about DataMemory and IndexMemory.

- The maximum number of operations that can be performed per transaction is set using the configuration parameters MaxNoOfConcurrentOperations and MaxNoOfLocalOperations. Note that bulk loading, TRUNCATE TABLE, and ALTER TABLE are handled as special cases by running multiple transactions, and so are not subject to this limitation.
- Different limits related to tables and indexes. For example, the maximum number of ordered indexes per table is determined by MaxNoOfOrderedIndexes.

- Database names, table names and attribute names cannot be as long in NDB tables as with other table handlers. Attribute names are truncated to 31 characters, and if not unique after truncation give rise to errors. Database names and table names can total a maximum of 122 characters. (That is, the maximum length for an NDB CLUSTER table

name is 122 characters less the number of characters in the name of the database of which that table is a part.)

- All Cluster table rows are of fixed length. This means (for example) that if a table has one or more VARCHAR fields containing only relatively small values, more memory and disk space is required when using the NDB storage engine than would be the case for the same table and data using the MyISAM engine. (In other words, in the case of a VARCHAR column, the column requires the same amount of storage as a CHAR column of the same size.)
- The maximum number of tables in a Cluster database is limited to 1792.
- The maximum number of ordered indexes per cluster, including AUTO_INCREMENT columns and hidden primary keys, is 2048.

This limitation was lifted in MySQL 5.0.23.

- The maximum number of attributes per table is limited to 128.
 - The maximum permitted size of any one row is 8KB. Note that each BLOB or TEXT column contributes a maximum of 256 bytes towards this total.
 - The maximum number of attributes per key is 32.
- **Unsupported features** (do not cause errors, but are not supported or enforced):
 - The foreign key construct is ignored, just as it is in MyISAM tables.
 - Savepoints and rollbacks to savepoints are ignored as in MyISAM.
 - OPTIMIZE operations are not supported.
 - LOAD TABLE ... FROM MASTER is not supported.
 - **Performance and limitation-related issues:**

- There are query performance issues due to sequential access to the NDB storage engine; it is also relatively more expensive to do many range scans than it is with either MyISAM or InnoDB.
- The `Records in range` statistic is not supported, resulting in non-optimal query plans in some cases. Employ `USE INDEX` or `FORCE INDEX` as a workaround.
- Unique hash indexes created with `USING HASH` cannot be used for accessing a table if `NULL` is given as part of the key.
- MySQL Cluster does not support durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.
- `SQL_LOG_BIN` has no effect on data operations; however, it is supported for schema operations.

MySQL Cluster cannot produce a binlog for tables having BLOB columns but no primary key.

Only the following schema operations are logged in a cluster binlog which is *not* on the **mysqld** executing the statement:

- `CREATE TABLE`
- `ALTER TABLE`
- `DROP TABLE`
- `CREATE DATABASE / CREATE SCHEMA`
- `DROP DATABASE / DROP SCHEMA`

- **Missing features:**

- The only supported isolation level is `READ COMMITTED`. (InnoDB supports `READ COMMITTED`, `READ UNCOMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`.) See [Section 15.8.5, “Backup Troubleshooting”](#), for information on how this can affect backup and restore of Cluster

databases.

- No durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.
- **Problems relating to multiple MySQL servers** (not relating to MyISAM or InnoDB):
 - ALTER TABLE is not fully locking when running multiple MySQL servers (no distributed table lock).
 - MySQL replication will not work correctly if updates are done on multiple MySQL servers. However, if the database partitioning scheme is done at the application level and no transactions take place across these partitions, replication can be made to work.
 - Autodiscovery of databases is not supported for multiple MySQL servers accessing the same MySQL Cluster. However, autodiscovery of tables is supported in such cases. What this means is that after a database named *db_name* is created or imported using one MySQL server, you should issue a CREATE DATABASE db_name statement on each additional MySQL server that accesses the same MySQL Cluster. (As of MySQL 5.0.2, you may also use CREATE SCHEMA db_name.) Once this has been done for a given MySQL server, that server should be able to detect the database tables without error.
 - DDL operations are not node failure safe. If a node fails while trying to perform one of these (such as CREATE TABLE or ALTER TABLE), the data dictionary is locked and no further DDL statements can be executed without restarting the cluster.
- **Issues exclusive to MySQL Cluster** (not related to MyISAM or InnoDB):
 - All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running **mysql** or other clients that may be accessing the cluster's SQL nodes.

- It is also not possible to perform a Cluster backup and restore between different architectures. For example, you cannot back up a cluster running on a big-endian platform and then restore from that backup to a cluster running on a little-endian system. (Bug #19255)
- It is not possible to make online schema changes such as those accomplished using `ALTER TABLE` or `CREATE INDEX`, as the NDB Cluster engine does not support autodiscovery of such changes. (However, you can import or create a table that uses a different storage engine, and then convert it to NDB using `ALTER TABLE tbl_name ENGINE=NDBCLUSTER`. In such a case, you must issue a `FLUSH TABLES` statement to force the cluster to pick up the change.)
- Online adding or dropping of nodes is not possible (the cluster must be restarted in such cases).
- When using multiple management servers:
 - You must give nodes explicit IDs in connectstrings because automatic allocation of node IDs does not work across multiple management servers.
 - You must take extreme care to have the same configurations for all management servers. No special checks for this are performed by the cluster.
 - Prior to MySQL 5.0.14, all data nodes had to be restarted after bringing up the cluster in order for the management nodes to be able to see one another.

(See Bug #12307 and #13070 for more information.)
- Multiple network interfaces for data nodes are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits for confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable.
- The maximum number of data nodes is 48.

- The total maximum number of nodes in a MySQL Cluster is 63. This number includes all MySQL Servers (SQL nodes), data nodes, and management servers.

The following Cluster limitations in MySQL 4.1 have been resolved in MySQL 5.0 as shown below:

- The NDB Cluster storage engine supports all character sets and collations available in MySQL 5.0.
- Prior to MySQL 5.0.6, the maximum number of metadata objects possible was 1600. Beginning with 5.0.6, this limit is increased to 20320.
- Cluster in MySQL 5.0 supports column indexes that make use of prefixes.
- Unlike the case in MySQL 4.1, the Cluster storage engine in MySQL 5.0 supports MySQL' query cache. See [Section 5.14, “The MySQL Query Cache”](#).
- Beginning with MySQL 5.0.21, it is possible to install MySQL with Cluster support to a non-default location and change the search path for font description files using either the `--basedir` or `--character-sets-dir` options. (Previously, **ndbd** in MySQL 5.0 searched only the default path — typically `/usr/local/mysql/share/mysql/charsets` — for character sets.)

15.11. MySQL Cluster Development Roadmap

In this section, we discuss changes in the implementation of MySQL Cluster in MySQL 5.0 as compared to MySQL 4.1. We will also discuss our roadmap for further improvements to MySQL Cluster as currently planned for MySQL 5.1.

There are relatively few changes between the NDB Cluster storage engine implementations in MySQL 4.1 and in 5.0, so the upgrade path should be relatively quick and painless.

All significantly new features being developed for MySQL Cluster are going into the MySQL 5.1 and 5.2 trees. For information on changes in the Cluster implementations in MySQL versions 5.1 and later, see <http://dev.mysql.com/doc/refman/5.1/en/ndbcluster.html>.

15.11.1. MySQL Cluster Changes in MySQL 5.0

MySQL Cluster in versions 5.0.3-beta and later contains a number of new features that are likely to be of interest:

- **Push-Down Conditions:** Consider the following query:

```
SELECT * FROM t1 WHERE non_indexed_attribute = 1;
```

This query will use a full table scan and the condition will be evaluated in the cluster's data nodes. Thus, it is not necessary to send the records across the network for evaluation. (That is, function transport is used, rather than data transport.) Please note that this feature is currently disabled by default (pending more thorough testing), but it should work in most cases. This feature can be enabled through the use of the `SET engine_condition_pushdown = on` statement. Alternatively, you can run **mysqld** with this feature enabled by starting the MySQL server with the `--engine-condition-pushdown` option.

A major benefit of this change is that queries can be executed in parallel. This means that queries against non-indexed columns can run faster than previously by a factor of as much as 5 to 10 times, *times the number of data nodes*, because multiple CPUs can work on the query in parallel.

You can use `EXPLAIN` to determine when condition pushdown is being used. See [Section 7.2.1, “Optimizing Queries with `EXPLAIN`”](#).

- **Decreased IndexMemory Usage:** In MySQL 5.0, each record consumes approximately 25 bytes of index memory, and every unique index uses 25 bytes per record of index memory (in addition to some data memory because these are stored in a separate table). This is because the primary key is not stored in the index memory anymore.
- **Query Cache Enabled for MySQL Cluster:** See [Section 5.14, “The MySQL Query Cache”](#), for information on configuring and using the query cache.
- **New Optimizations:** One optimization that merits particular attention is that a batched read interface is now used in some queries. For example, consider the following query:

```
SELECT * FROM t1 WHERE primary_key IN (1,2,3,4,5,6,7,8,9,10);
```

This query will be executed 2 to 3 times more quickly than in previous MySQL Cluster versions due to the fact that all 10 key lookups are sent in a single batch rather than one at a time.

- **Limit On Number of Metadata Objects:** Beginning with MySQL 5.0.6, each Cluster database may contain a maximum of 20320 metadata objects — this includes database tables, system tables, indexes and BLOB values. (Previously, this number was 1600.)

15.11.2. MySQL 5.1 Development Roadmap for MySQL Cluster

What is said here is a status report based on recent commits to the MySQL 5.1 source tree. It should be noted all 5.1 development is subject to change.

There are currently 4 major new features being developed for MySQL 5.1:

1. **Integration of MySQL Cluster into MySQL replication:** This will make it possible to update from any MySQL Server in the cluster and still have the MySQL Replication handled by one of the MySQL Servers in the cluster, with the state of the slave side remaining consistent with the cluster acting as the master.

2. **Support for disk-based records:** Records on disk will be supported. Indexed fields including the primary key hash index must still be stored in RAM but all other fields can be on disk.
3. **Variable-sized records:** A column defined as `VARCHAR(255)` currently uses 260 bytes of storage independent of what is stored in any particular record. In MySQL 5.1 Cluster tables, only the portion of the column actually taken up by the record will be stored. This will make possible a reduction in space requirements for such columns by a factor of 5 in many cases.
4. **User-defined partitioning:** Users will be able to define partitions based on columns that are part of the primary key. The MySQL Server will be able to discover whether it is possible to prune away some of the partitions from the `WHERE` clause. Partitioning based on `KEY`, `HASH`, `RANGE`, and `LIST` handlers will be possible, as well as subpartitioning. This feature should also be available for many other handlers, and not only `NDB Cluster`.

In addition, we are working to increase the 8KB size limit for rows containing columns of types other than `BLOB` or `TEXT` in Cluster tables. This is due to the fact that rows are currently fixed in size and the page size is 32,768 bytes (minus 128 bytes for the row header). Currently, this means that if we allowed more than 8KB per record, any remaining space (up to approximately 14,000 bytes) would be left empty. In MySQL 5.1, we plan to fix this limitation so that using more than 8KB in a given row does not result in the remainder of the page being wasted.

15.12. MySQL Cluster FAQ

This section answers questions that are often asked about MySQL Cluster.

- *What does “NDB” mean?*

This stands for “**N**etwork **D**atabase.”

- *What's the difference in using Cluster vs using replication?*

In a replication setup, a master MySQL server updates one or more slaves. Transactions are committed sequentially, and a slow transaction can cause the slave to lag behind the master. This means that if the master fails, it is possible that the slave might not have recorded the last few transactions. If a transaction-safe engine such as InnoDB is being used, a transaction will either be complete on the slave or not applied at all, but replication does not guarantee that all data on the master and the slave will be consistent at all times. In MySQL Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is asynchronous, MySQL Cluster is synchronous.

We have implemented (asynchronous) replication for Cluster in MySQL 5.1. This includes the capability to replicate both between two clusters, and from a MySQL cluster to a non-Cluster MySQL server. However, we do not plan to backport this functionality to MySQL 5.0.

- *Do I need to do any special networking to run Cluster? (How do computers in a cluster communicate?)*

MySQL Cluster is intended to be used in a high-bandwidth environment, with computers connecting via TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for Cluster include a typical 100-megabit Ethernet network or the equivalent. We recommend you use gigabit

Ethernet whenever available.

The faster SCI protocol is also supported, but requires special hardware. See [Section 15.9, “Using High-Speed Interconnects with MySQL Cluster”](#), for more information about SCI.

- *How many computers do I need to run a cluster, and why?*

A minimum of three computers is required to run a viable cluster. However, the minimum **recommended** number of computers in a MySQL Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

- *What do the different computers do in a cluster?*

A MySQL Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as *nodes*, and a computer housing a cluster node is sometimes referred to as a *cluster host*. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- **Management node (MGM node):** Provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application **ndb_mgmd**; the management client used to control MySQL Cluster via the MGM node is **ndb_mgm**.
 - **Data node:** Stores and replicates data. Data node functionality is handled by an instance of the NDB data node process **ndbd**.
 - **SQL node:** This is simply an instance of MySQL Server (**mysqld**) that is built with support for the NDB Cluster storage engine and started with the **--ndb-cluster** option to enable the engine.
- *With which operating systems can I use Cluster?*

MySQL Cluster is officially supported on Linux, Mac OS X, and Solaris.

We are working to add Cluster support for other platforms, including Windows, and our goal is eventually to offer MySQL Cluster on all platforms for which MySQL itself is supported.

It may be possible to run Cluster processes on other operating systems. We have had reports from users who say that they have run Cluster successfully on FreeBSD as well as HP-UX. However, Cluster on any but the three platforms mentioned here should be considered alpha software (at best), cannot be guaranteed reliable in a production setting, and *is not supported by MySQL AB*.

- *What are the hardware requirements for running MySQL Cluster?*

Cluster should run on any platform for which NDB-enabled binaries are available. Naturally, faster CPUs and more memory will improve performance, and 64-bit CPUs will likely be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see *How much RAM do I Need?* for more information). Nodes can communicate via a standard TCP/IP network and hardware. For SCI support, special networking hardware is required.

- *How much RAM do I need? Is it possible to use disk memory at all?*

In MySQL-5.0, Cluster is in-memory only. This means that all table data (including indexes) is stored in RAM. Therefore, if your data takes up 1GB of space and you want to replicate it once in the cluster, you need 2GB of memory to do so. This is in addition to the memory required by the operating system and any applications running on the cluster computers.

If a data node's memory usage exceeds what is available in RAM, then the system will attempt to use swap space up to the limit set for `DataMemory`. However, this will at best result in severely degraded performance, and may cause the node to be dropped due to slow response time (missed heartbeats). We do not recommend on relying on disk swapping in a production environment for this reason. In any case, once the `DataMemory` limit is reached, any operations requiring additional memory (such as inserts) will fail.

(We have implemented disk data storage for MySQL Cluster in MySQL

5.1, but we have no plans to add this capability in MySQL 5.0.)

You can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

$$(\text{SizeofDatabase} \times \text{NumberOfReplicas} \times 1.1) / \text{NumberOfDataNodes}$$

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see [Section 11.5, “Data Type Storage Requirements”](#), for details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an NDBCluster table requires 21–25 bytes per record. These indexes use `IndexMemory`.
- Each ordered index requires 10 bytes storage per record, using `DataMemory`.
- Creating a primary key or unique index also creates an ordered index, unless this index is created with `USING HASH`. In other words, if created without `USING HASH`, a primary key or unique index on a Cluster table takes up 31–35 bytes per record in MySQL 5.0.

Note that creating MySQL Cluster tables with `USING HASH` for all primary keys and unique indexes will generally cause table updates to run more quickly. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated).

When calculating Cluster memory requirements, you may find useful the `ndb_size.pl` utility which is available on [MySQLForge](#). This Perl script connects to a current MySQL (non-Cluster) database and creates a report on how much space that database would require if it used the NDBCluster storage engine.

It is especially important to keep in mind that *every MySQL Cluster table must have a primary key*. The NDB storage engine creates a primary key automatically if none is defined, and this primary key is created without `USING HASH`.

There is no easy way to determine exactly how much memory is being used for storage of Cluster indexes at any given time; however, warnings are written to the Cluster log when 80% of available DataMemory or IndexMemory is in use, and again when use reaches 85%, 90%, and so on.

We often see questions from users who report that, when they are trying to populate a Cluster database, the loading process terminates prematurely and an error message like this one is observed:

```
ERROR 1114: The table 'my_cluster_table' is full
```

When this occurs, the cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, *including the primary key required by the NDB storage engine and automatically created in the event that the table definition does not include the definition of a primary key.*

It is also worth noting that all data nodes should have the same amount of RAM, as no data node in a cluster can use more memory than the least amount available to any individual data node. In other words, if there are three computers hosting Cluster data nodes, with two of these having 3GB of RAM available to store Cluster data, and one having only 1GB RAM, then each data node can devote only 1GB to clustering.

- *Because MySQL Cluster uses TCP/IP, does that mean I can run it over the Internet, with one or more nodes in a remote location?*

It is very doubtful in any case that a cluster would perform reliably under such conditions, as MySQL Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet (preferably the latter). We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in a MySQL Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you should take the same precautions as you

would with any other instance of the MySQL server.)

- *Do I have to learn a new programming or query language to use Cluster?*

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL queries and commands are required for the following operations:

- Creating, altering, and dropping tables
 - Inserting, updating, and deleting table data
 - Creating, changing, and dropping primary and unique indexes
 - Configuring and managing SQL nodes (MySQL servers)
- *How do I find out what an error or warning message means when using Cluster?*

There are two ways in which this can be done:

- From within the **mysql** client, use **SHOW ERRORS** or **SHOW WARNINGS** immediately upon being notified of the error or warning condition. Errors and warnings also be displayed in MySQL Query Browser.
 - From a system shell prompt, use **perror --ndb error_code**.
- *Is MySQL Cluster transaction-safe? What isolation levels are supported?*

Yes: For tables created with the NDB storage engine, transactions are supported. In MySQL 5.0, Cluster supports only the READ COMMITTED transaction isolation level.

- *What storage engines are supported by MySQL Cluster?*

Clustering in MySQL is supported only by the NDB storage engine. That is, in order for a table to be shared between nodes in a cluster, it must be created using ENGINE=NDB (or ENGINE=NDBCLUSTER, which is equivalent).

(It is possible to create tables using other storage engines such as MyISAM or

InnoDB on a MySQL server being used for clustering, but these non-NDB tables will **not** participate in the cluster.)

- *Which versions of the MySQL software support Cluster? Do I have to compile from source?*

Cluster is supported in all MySQL-max binaries in the 5.0 release series, except as noted in the following paragraph. You can determine whether your server has NDB support using either the `SHOW VARIABLES LIKE 'have_%'` or `SHOW ENGINES` statement. (See [Section 5.3, “The `mysqld-max` Extended MySQL Server”](#), for more information.)

Linux users, please note that NDB is *not* included in the standard MySQL server RPMs. Beginning with MySQL 5.0.4, there are separate RPM packages for the NDB storage engine and accompanying management and other tools; see the NDB RPM Downloads section of the MySQL 5.0 Downloads page for these. (Prior to 5.0.4, you had to use the -max binaries supplied as `.tar.gz` archives. This is still possible, but is not required, so you can use your Linux distribution's RPM manager if you prefer.) You can also obtain NDB support by compiling the -max binaries from source, but it is not necessary to do so simply to use MySQL Cluster. To download the latest binary, RPM, or source distribution in the MySQL 5.0 series, visit <http://dev.mysql.com/downloads/mysql/5.0.html>.

- *In the event of a catastrophic failure — say, for instance, the whole city loses power **and** my UPS fails — would I lose all my data?*

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

- *Is it possible to use `FULLTEXT` indexes with Cluster?*

`FULLTEXT` indexing is not currently supported by the NDB storage engine, or by any storage engine other than `MyISAM`. We are working to add this capability in a future release.

- *Can I run multiple nodes on a single computer?*

It is possible but not advisable. One of the chief reasons to run a cluster is to provide redundancy. To enjoy the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. Given that MySQL Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 200 MHz Pentium CPU and sufficient RAM for the operating system plus a small amount of overhead for the **ndb_mgmd** and **ndb_mgm** processes.

It is acceptable to run multiple cluster data nodes on a single host for learning about MySQL Cluster, or for testing purposes; however, this is not supported for production use.

- *Can I add nodes to a cluster without restarting it?*

Not at present. A simple restart is all that is required for adding new MGM or SQL nodes to a Cluster. When adding data nodes the process is more complex, and requires the following steps:

1. Make a complete backup of all Cluster data.
2. Completely shut down the cluster and all cluster node processes.
3. Restart the cluster, using the `--initial` startup option.
4. Restore all cluster data from the backup.

In a future MySQL Cluster release series, we hope to implement a “hot” reconfiguration capability for MySQL Cluster to minimize (if not eliminate) the requirement for restarting the cluster when adding new nodes.

- *Are there any limitations that I should be aware of when using Cluster?*

NDB tables in MySQL are subject to the following limitations:

- Not all character sets and collations are supported.

- FULLTEXT indexes and index prefixes are not supported. Only complete columns may be indexed.
- Spatial data types are not supported. See [Chapter 16, *Spatial Extensions*](#).
- Only complete rollbacks for transactions are supported. Partial rollbacks and rollbacks to savepoints are not supported.
- The maximum number of attributes allowed per table is 128, and attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- The maximum size for a table row is 8 kilobytes, not counting BLOB values. There is no set limit for the number of rows per table. Table size limits depend on a number of factors, in particular on the amount of RAM available to each data node.
- The NDB engine does not support foreign key constraints. As with MyISAM tables, these are ignored.
- Query caching is not supported.

For additional information on Cluster limitations, see [Section 15.10, “Known Limitations of MySQL Cluster”](#).

- *How do I import an existing MySQL database into a cluster?*

You can import databases into MySQL Cluster much as you would with any other version of MySQL. Other than the limitation mentioned in the previous question, the only other special requirement is that any tables to be included in the cluster must use the NDB storage engine. This means that the tables must be created with ENGINE=NDB or ENGINE=NDBCLUSTER. It is also possible to convert existing tables using other storage engines to NDB Cluster using ALTER TABLE, but requires an additional workaround. See [Section 15.10, “Known Limitations of MySQL Cluster”](#), for details.

- *How do cluster nodes communicate with one another?*

Cluster nodes can communicate via any of three different protocols: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See [Section 15.9, “Using High-Speed Interconnects with MySQL Cluster”](#), for more about using SCI as a transport mechanism in MySQL Cluster.

- *What is an “arbitrator”?*

If one or more nodes in a cluster fail, it is possible that not all cluster nodes will be able to “see” one another. In fact, it is possible that two sets of nodes might become isolated from one another in a network partitioning, also known as a “split brain” scenario. This type of situation is undesirable because each set of nodes tries to behave as though it is the entire cluster.

When cluster nodes go down, there are two possibilities. If more than 50% of the remaining nodes can communicate with each other, we have what is sometimes called a “majority rules” situation, and this set of nodes is considered to be the cluster. The arbitrator comes into play when there is an even number of nodes: in such cases, the set of nodes to which the arbitrator belongs is considered to be the cluster, and nodes not belonging to this set are shut down.

The preceding information is somewhat simplified. A more complete explanation taking into account node groups follows:

When all nodes in at least one node group are alive, network partitioning is not an issue, because no one portion of the cluster can form a functional cluster. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the “split-brain” scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognize the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbitrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbitrator selection is controlled by the `ArbitrationRank` configuration parameter for MySQL Server and management server nodes.

(See [Section 15.4.4.4, “Defining the Management Server”](#), for details.) It should also be noted that the role of arbitrator does not in and of itself impose any heavy demands upon the host so designated, and thus the arbitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

- *What data types are supported by MySQL Cluster?*

MySQL Cluster supports all of the usual MySQL data types, with the exception of those associated with MySQL's spatial extensions. (See [Chapter 16, *Spatial Extensions*](#).) In addition, there are some differences with regard to indexes when used with NDB tables. **Note:** MySQL Cluster tables (that is, tables created with `ENGINE=NDBCLUSTER`) have only fixed-width rows. This means that (for example) each record containing a `VARCHAR(255)` column will require space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein. This issue is expected to be fixed in a future MySQL release series.

See [Section 15.10, “Known Limitations of MySQL Cluster”](#), for more information about these issues.

- *How do I start and stop MySQL Cluster?*

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node with the **`ndb_mgmd`** command.
2. Start each data node with the **`ndbd`** command.
3. Start each MySQL server (SQL node) using **`mysqld_safe --user=mysql &`**.

Each of these commands must be run from a system shell on the machine housing the affected node. You can verify the cluster is running by starting the MGM management client **`ndb_mgm`** on the machine housing the MGM node.

- *What happens to cluster data when the cluster is shut down?*

The data held in memory by the cluster's data nodes is written to disk, and is reloaded in memory the next time that the cluster is started.

To shut down the cluster, enter the following command in a shell on the machine hosting the MGM node:

```
shell> ndb_mgm -e shutdown
```

This causes the **ndb_mgm**, **ndb_mgm**, and any **ndbd** processes to terminate gracefully. MySQL servers running as Cluster SQL nodes can be stopped using **mysqladmin shutdown**.

For more information, see [Section 15.7.2, “Commands in the Management Client”](#), and [Section 15.3.6, “Safe Shutdown and Restart”](#).

- *Is it helpful to have more than one management node for a cluster?*

It can be helpful as a fail-safe. Only one MGM node controls the cluster at any given time, but it is possible to configure one MGM as primary, and one or more additional management nodes to take over in the event that the primary MGM node fails.

- *Can I mix different kinds of hardware and operating systems in a Cluster?*

Yes, so long as all machines and operating systems have the same endianness (all big-endian or all little-endian). It is also possible to use different MySQL Cluster releases on different nodes. However, we recommend this be done only as part of a rolling upgrade procedure.

- *Can I run two data nodes on a single host? Two SQL nodes?*

Yes, it is possible to do this. In the case of multiple data nodes, each node must use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of **mysqld** must use a different TCP/IP port. However, running more than one node of a given type per machine is not supported for production use.

- *Can I use hostnames with MySQL Cluster?*

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your

application requires “five nines” availability, we recommend using fixed IP addresses. Making communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional points of failure, and the fewer of these, the better.

15.13. MySQL Cluster Glossary

The following terms are useful to an understanding of MySQL Cluster or have specialized meanings when used in relation to it.

- **Cluster:**

In its generic sense, a cluster is a set of computers functioning as a unit and working together to accomplish a single task.

NDB Cluster:

This is the storage engine used in MySQL to implement data storage, retrieval, and management distributed among several computers.

MySQL Cluster:

This refers to a group of computers working together using the NDB storage engine to support a distributed MySQL database in a *shared-nothing architecture* using *in-memory storage*.

- **Configuration files:**

Text files containing directives and information regarding the cluster, its hosts, and its nodes. These are read by the cluster's management nodes when the cluster is started. See [Section 15.4.4, “Configuration File”](#), for details.

- **Backup:**

A complete copy of all cluster data, transactions and logs, saved to disk or other long-term storage.

- **Restore:**

Returning the cluster to a previous state, as stored in a backup.

- **Checkpoint:**

Generally speaking, when data is saved to disk, it is said that a checkpoint has been reached. More specific to Cluster, it is a point in time where all committed transactions are stored on disk. With regard to the NDB storage engine, there are two types of checkpoints which work together to ensure that a consistent view of the cluster's data is maintained:

- **Local Checkpoint (LCP):**

This is a checkpoint that is specific to a single node; however, LCP's take place for all nodes in the cluster more or less concurrently. An LCP involves saving all of a node's data to disk, and so usually occurs every few minutes. The precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.

- **Global Checkpoint (GCP):**

A GCP occurs every few seconds, when transactions for all nodes are synchronized and the redo-log is flushed to disk.

- **Cluster host:**

A computer making up part of a MySQL Cluster. A cluster has both a *physical* structure and a *logical* structure. Physically, the cluster consists of a number of computers, known as *cluster hosts* (or more simply as *hosts*). See also **Node** and **Node group** below.

- **Node:**

This refers to a logical or functional unit of MySQL Cluster, and is sometimes also referred to as a *cluster node*. In the context of MySQL Cluster, we use the term “node” to indicate a *process* rather than a physical component of the cluster. There are three node types required to implement a working MySQL Cluster:

- **Management (MGM) nodes:**

Manages the other nodes within the MySQL Cluster. It provides configuration data to the other nodes; starts and stops nodes; handles network partitioning; creates backups and restores from them, and so

forth.

- **SQL (MySQL server) nodes:**

Instances of MySQL Server which serve as front ends to data kept in the cluster's **data nodes**. Clients desiring to store, retrieve, or update data can access an SQL node just as they would any other MySQL Server, employing the usual authentication methods and APIs; the underlying distribution of data between node groups is transparent to users and applications. SQL nodes access the cluster's databases as a whole without regard to the data's distribution across different data nodes or cluster hosts.

- **Data nodes:**

These nodes store the actual data. Table data fragments are stored in a set of node groups; each node group stores a different subset of the table data. Each of the nodes making up a node group stores a replica of the fragment for which that node group is responsible. Currently, a single cluster can support up to 48 data nodes total.

It is possible for more than one node to co-exist on a single machine. (In fact, it is even possible to set up a complete cluster on one machine, although one would almost certainly *not* want to do this in a production environment.) It may be helpful to remember that, when working with MySQL Cluster, the term *host* refers to a physical component of the cluster whereas a *node* is a logical or functional component (that is, a process).

Note Regarding Terms: In older versions of the MySQL Cluster documentation, data nodes were sometimes referred to as “database nodes”. The term “storage nodes” has also been used. In addition, SQL nodes were sometimes known as “client nodes”. They are also often referred to as “API nodes”. The older terminology has been deprecated to minimize confusion, and for this reason should be avoided.

- **Node group:**

A set of data nodes. All data nodes in a node group contain the same data (fragments), and all nodes in a single group should reside on different hosts. It is possible to control which nodes belong to which node groups.

For more information, see [Section 15.2.1, “MySQL Cluster Nodes, Node Groups, Replicas, and Partitions”](#).

- **Node failure:**

MySQL Cluster is not solely dependent upon the functioning of any single node making up the cluster; the cluster can continue to run if one or more nodes fail. The precise number of node failures that a given cluster can tolerate depends upon the number of nodes and the cluster's configuration.

- **Node restart:**

The process of restarting a failed cluster node.

- **Initial node restart:**

The process of starting a cluster node with its filesystem removed. This is sometimes used in the course of software upgrades and in other special circumstances.

- **System crash (or system failure):**

This can occur when so many cluster nodes have failed that the cluster's state can no longer be guaranteed.

- **System restart:**

The process of restarting the cluster and reinitializing its state from disk logs and checkpoints. This is required after either a planned or an unplanned shutdown of the cluster.

- **Fragment:**

A portion of a database table; in the NDB storage engine, a table is broken up into and stored as a number of fragments. A fragment is sometimes also called a “partition”; however, “fragment” is the preferred term. Tables are fragmented in MySQL Cluster in order to facilitate load balancing between machines and nodes.

- **Replica:**

Under the NDB storage engine, each table fragment has number of replicas stored on other data nodes in order to provide redundancy. Currently, there may be up four replicas per fragment.

- **Transporter:**

A protocol providing data transfer between nodes. MySQL Cluster currently supports four different types of transporter connections:

- TCP/IP

This is, of course, the familiar network protocol that underlies HTTP, FTP (and so on) on the Internet. TCP/IP can be used for both local and remote connections.

- SCI

Scalable Coherent Interface is a high-speed protocol used in building multiprocessor systems and parallel-processing applications. Use of SCI with MySQL Cluster requires specialized hardware, as discussed in [Section 15.9.1, “Configuring MySQL Cluster to use SCI Sockets”](#). For a basic introduction to SCI, see [this essay at dolphinics.com](#).

- SHM

Unix-style **shared memory** segments. Where supported, SHM is used automatically to connect nodes running on the same host. The [Unix man page for shmop\(2\)](#) is a good place to begin obtaining additional information about this topic.

Note: The cluster transporter is internal to the cluster. Applications using MySQL Cluster communicate with SQL nodes just as they do with any other version of MySQL Server (via TCP/IP, or through the use of Unix socket files or Windows named pipes). Queries can be sent and results retrieved using the standard MySQL client APIs.

- **NDB:**

This stands for **Network Database**, and refers to the storage engine used to enable MySQL Cluster. The NDB storage engine supports all the usual

MySQL data types and SQL statements, and is ACID-compliant. This engine also provides full support for transactions (commits and rollbacks).

- **shared-nothing architecture:**

The ideal architecture for a MySQL Cluster. In a true shared-nothing setup, each node runs on a separate host. The advantage such an arrangement is that there no single host or node can act as single point of failure or as a performance bottle neck for the system as a whole.

- **In-memory storage:**

All data stored in each data node is kept in memory on the node's host computer. For each data node in the cluster, you must have available an amount of RAM equal to the size of the database times the number of replicas, divided by the number of data nodes. Thus, if the database takes up 1GB of memory, and you want to set up the cluster with four replicas and eight data nodes, a minimum of 500MB memory will be required per node. Note that this is in addition to any requirements for the operating system and any other applications that might be running on the host.

- **Table:**

As is usual in the context of a relational database, the term “table” denotes a set of identically structured records. In MySQL Cluster, a database table is stored in a data node as a set of fragments, each of which is replicated on additional data nodes. The set of data nodes replicating the same fragment or set of fragments is referred to as a *node group*.

- **Cluster programs:**

These are command-line programs used in running, configuring, and administering MySQL Cluster. They include both server daemons:

- **ndbd:**

The data node daemon (runs a data node process)

- **ndb_mgmd:**

The management server daemon (runs a management server process)

and client programs:

- **ndb_mgm:**

The management client (provides an interface for executing management commands)

- **ndb_waiter:**

Used to verify status of all nodes in a cluster

- **ndb_restore:**

Restores cluster data from backup

For more about these programs and their uses, see [Section 15.6, “Process Management in MySQL Cluster”](#).

- **Event log:**

MySQL Cluster logs events by category (startup, shutdown, errors, checkpoints, and so on), priority, and severity. A complete listing of all reportable events may be found in [Section 15.7.3, “Event Reports Generated in MySQL Cluster”](#). Event logs are of two types:

- **Cluster log:**

Keeps a record of all desired reportable events for the cluster as a whole.

- **Node log:**

A separate log which is also kept for each individual node.

Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be consulted only for application development and debugging purposes.

Chapter 16. Spatial Extensions

Table of Contents

[16.1. Introduction to MySQL Spatial Support](#)

[16.2. The OpenGIS Geometry Model](#)

[16.2.1. The Geometry Class Hierarchy](#)

[16.2.2. Class Geometry](#)

[16.2.3. Class Point](#)

[16.2.4. Class Curve](#)

[16.2.5. Class LineString](#)

[16.2.6. Class Surface](#)

[16.2.7. Class Polygon](#)

[16.2.8. Class GeometryCollection](#)

[16.2.9. Class MultiPoint](#)

[16.2.10. Class MultiCurve](#)

[16.2.11. Class MultiLineString](#)

[16.2.12. Class MultiSurface](#)

[16.2.13. Class MultiPolygon](#)

[16.3. Supported Spatial Data Formats](#)

[16.3.1. Well-Known Text \(WKT\) Format](#)

[16.3.2. Well-Known Binary \(WKB\) Format](#)

[16.4. Creating a Spatially Enabled MySQL Database](#)

[16.4.1. MySQL Spatial Data Types](#)

[16.4.2. Creating Spatial Values](#)

[16.4.3. Creating Spatial Columns](#)

[16.4.4. Populating Spatial Columns](#)

[16.4.5. Fetching Spatial Data](#)

[16.5. Analyzing Spatial Information](#)

[16.5.1. Geometry Format Conversion Functions](#)

[16.5.2. Geometry Functions](#)

[16.5.3. Functions That Create New Geometries from Existing Ones](#)

[16.5.4. Functions for Testing Spatial Relations Between Geometric Objects](#)

[16.5.5. Relations on Geometry Minimal Bounding Rectangles \(MBRs\)](#)

[16.5.6. Functions That Test Spatial Relationships Between Geometries](#)

[16.6. Optimizing Spatial Analysis](#)

[16.6.1. Creating Spatial Indexes](#)

[16.6.2. Using a Spatial Index](#)

[16.7. MySQL Conformance and Compatibility](#)

MySQL supports spatial extensions to allow the generation, storage, and analysis of geographic features. Before MySQL 5.0.16, these features are available for MyISAM tables only. As of MySQL 5.0.16, InnoDB, NDB, BDB, and ARCHIVE also support spatial features. (However, the ARCHIVE engine does not support indexing, so spatial columns in ARCHIVE columns cannot be indexed. MySQL Cluster also does not support indexing of spatial columns.)

Although spatial extensions are supported in InnoDB tables, use of spatial indexes may cause a crash. (Bug #15860)

This chapter covers the following topics:

- The basis of these spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

Additional resources

- The Open Geospatial Consortium publishes the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at <http://www.opengis.org/docs/99-049.pdf>.
- If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: <http://forums.mysql.com/list.php?23>.

16.1. Introduction to MySQL Spatial Support

MySQL implements spatial extensions following the specification of the Open Geospatial Consortium (OGC). This is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. The OGC maintains a Web site at <http://www.opengis.org/>.

In 1997, the Open Geospatial Consortium published the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at <http://www.opengis.org/docs/99-049.pdf>. It contains additional information relevant to this chapter.

MySQL implements a subset of the **SQL with Geometry Types** environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describe a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, a postcode area, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

This chapter uses all of these terms synonymously: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. Here, the term most commonly used is **geometry**, defined as *a point or an aggregate of points representing anything in the world that has a location*.

16.2. The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

16.2.1. The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- Geometry (non-instantiable)
 - Point (instantiable)
 - Curve (non-instantiable)
 - LineString (instantiable)
 - Line
 - LinearRing
 - Surface (non-instantiable)
 - Polygon (instantiable)
 - GeometryCollection (instantiable)
 - MultiPoint (instantiable)
 - MultiCurve (non-instantiable)
 - MultiLineString (instantiable)

- MultiSurface (non-instantiable)
 - MultiPolygon (instantiable)

It is not possible to create objects in non-instantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

Geometry is the base class. It is an abstract class. The instantiable subclasses of Geometry are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base Geometry class has subclasses for Point, Curve, Surface, and GeometryCollection:

- Point represents zero-dimensional objects.
- Curve represents one-dimensional objects, and has subclass LineString, with sub-subclasses Line and LinearRing.
- Surface is designed for two-dimensional objects and has subclass Polygon.
- GeometryCollection has specialized zero-, one-, and two-dimensional collection classes named MultiPoint, MultiLineString, and MultiPolygon for modeling geometries corresponding to collections of Points, LineStrings, and Polygons, respectively. MultiCurve and MultiSurface are introduced as abstract superclasses that generalize the collection interfaces to handle Curves and Surfaces.

Geometry, Curve, Surface, MultiCurve, and MultiSurface are defined as non-instantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiLineString, and MultiPolygon are instantiable classes.

16.2.2. Class Geometry

Geometry is the root class of the hierarchy. It is a non-instantiable class but has a number of properties that are common to all geometry values created from any of the Geometry subclasses. These properties are described in the following list. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its **SRID**, or Spatial Reference Identifier. This value identifies the geometry's associated Spatial Reference System that describes the coordinate space in which the geometry object is defined.

In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

- Its **coordinates** in its Spatial Reference System, represented as double-precision (eight-byte) numbers. All non-empty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geocentric** system (coordinates on the Earth's surface) are different things.

- Its **interior**, **boundary**, and **exterior**.

Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.

- Its **MBR** (Minimum Bounding Rectangle), or Envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))

- Whether the value is **simple** or **non-simple**. Geometry values of types (`LineString`, `MultiPoint`, `MultiLineString`) are either simple or non-simple. Each type determines its own assertions for being simple or non-simple.
- Whether the value is **closed** or **not closed**. Geometry values of types (`LineString`, `MultiString`) are either closed or not closed. Each type determines its own assertions for being closed or not closed.
- Whether the value is **empty** or **non-empty** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a `NULL` value). An empty geometry is defined to be always simple and has an area of 0.
- Its **dimension**. A geometry can have a dimension of -1 , 0 , 1 , or 2 :
 - -1 for an empty geometry.
 - 0 for a geometry with no length and no area.
 - 1 for a geometry with non-zero length and zero area.
 - 2 for a geometry with non-zero area.

Point objects have a dimension of zero. `LineString` objects have a dimension of 1. `Polygon` objects have a dimension of 2. The dimensions of `MultiPoint`, `MultiLineString`, and `MultiPolygon` objects are the same as the dimensions of the elements they consist of.

16.2.3. Class Point

A `Point` is a geometry that represents a single location in coordinate space.

Point Examples

- Imagine a large-scale map of the world with many cities. A `Point` object could represent each city.

- On a city map, a `Point` object could represent a bus stop.

Point Properties

- X-coordinate value.
- Y-coordinate value.
- `Point` is defined as a zero-dimensional geometry.
- The boundary of a `Point` is the empty set.

16.2.4. Class `Curve`

A `Curve` is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of `Curve` define the type of interpolation between points. `Curve` is a non-instantiable class.

curve Properties

- A `Curve` has the coordinates of its points.
- A `Curve` is defined as a one-dimensional geometry.
- A `Curve` is simple if it does not pass through the same point twice.
- A `Curve` is closed if its start point is equal to its endpoint.
- The boundary of a closed `Curve` is empty.
- The boundary of a non-closed `Curve` consists of its two endpoints.
- A `Curve` that is simple and closed is a `LinearRing`.

16.2.5. Class `LineString`

A `LineString` is a `Curve` with linear interpolation between points.

LineString Examples

- On a world map, `LineString` objects could represent rivers.
- In a city map, `LineString` objects could represent streets.

LineString Properties

- A `LineString` has coordinates of segments, defined by each consecutive pair of points.
- A `LineString` is a `Line` if it consists of exactly two points.
- A `LineString` is a `LinearRing` if it is both closed and simple.

16.2.6. Class Surface

A `Surface` is a two-dimensional geometry. It is a non-instantiable class. Its only instantiable subclass is `Polygon`.

Surface Properties

- A `Surface` is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple `Surface` as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple `Surface` is the set of closed curves corresponding to its exterior and interior boundaries.

16.2.7. Class Polygon

A `Polygon` is a planar `Surface` representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the `Polygon`.

Polygon Examples

- On a region map, `Polygon` objects could represent forests, districts, and so on.

Polygon Assertions

- The boundary of a Polygon consists of a set of LinearRing objects (that is, LineString objects that are both simple and closed) that make up its exterior and interior boundaries.
- A Polygon has no rings that cross. The rings in the boundary of a Polygon may intersect at a Point, but only as a tangent.
- A Polygon has no lines, spikes, or punctures.
- A Polygon has an interior that is a connected point set.
- A Polygon may have holes. The exterior of a Polygon with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a Polygon a simple geometry.

16.2.8. Class GeometryCollection

A GeometryCollection is a geometry that is a collection of one or more geometries of any class.

All the elements in a GeometryCollection must be in the same Spatial Reference System (that is, in the same coordinate system). There are no other constraints on the elements of a GeometryCollection, although the subclasses of GeometryCollection described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a MultiPoint may contain only Point elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

16.2.9. Class MultiPoint

A MultiPoint is a geometry collection composed of Point elements. The points are not connected or ordered in any way.

MultiPoint Examples

- On a world map, a MultiPoint could represent a chain of small islands.
- On a city map, a MultiPoint could represent the outlets for a ticket office.

MultiPoint Properties

- A MultiPoint is a zero-dimensional geometry.
- A MultiPoint is simple if no two of its Point values are equal (have identical coordinate values).
- The boundary of a MultiPoint is the empty set.

16.2.10. Class MultiCurve

A MultiCurve is a geometry collection composed of Curve elements. MultiCurve is a non-instantiable class.

MultiCurve Properties

- A MultiCurve is a one-dimensional geometry.
- A MultiCurve is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A MultiCurve boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a MultiCurve if it is in the boundaries of an odd number of MultiCurve elements.
- A MultiCurve is closed if all of its elements are closed.
- The boundary of a closed MultiCurve is always empty.

16.2.11. Class MultiLineString

A MultiLineString is a MultiCurve geometry collection composed of

LineString elements.

MultiLineString Examples

- On a region map, a MultiLineString could represent a river system or a highway system.

16.2.12. Class MultiSurface

A MultiSurface is a geometry collection composed of surface elements. MultiSurface is a non-instantiable class. Its only instantiable subclass is MultiPolygon.

MultiSurface Assertions

- Two MultiSurface surfaces have no interiors that intersect.
- Two MultiSurface elements have boundaries that intersect at most at a finite number of points.

16.2.13. Class MultiPolygon

A MultiPolygon is a MultiSurface object composed of Polygon elements.

MultiPolygon Examples

- On a region map, a MultiPolygon could represent a system of lakes.

MultiPolygon Assertions

- A MultiPolygon has no two Polygon elements with interiors that intersect.
- A MultiPolygon has no two Polygon elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A MultiPolygon may not have cut lines, spikes, or punctures. A MultiPolygon is a regular, closed point set.
- A MultiPolygon that has more than one Polygon has an interior that is not

connected. The number of connected components of the interior of a `MultiPolygon` is equal to the number of `Polygon` values in the `MultiPolygon`.

MultiPolygon Properties

- A `MultiPolygon` is a two-dimensional geometry.
- A `MultiPolygon` boundary is a set of closed curves (`LineString` values) corresponding to the boundaries of its `Polygon` elements.
- Each curve in the boundary of the `MultiPolygon` is in the boundary of exactly one `Polygon` element.
- Every curve in the boundary of an `Polygon` element is in the boundary of the `MultiPolygon`.

16.3. Supported Spatial Data Formats

This section describes the standard spatial data formats that are used to represent geometry objects in queries. They are:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format.

16.3.1. Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form.

Examples of WKT representations of geometry objects:

- A Point:

```
POINT(15 20)
```

Note that point coordinates are specified with no separating comma.

- A LineString with four points:

```
LINestring(0 0, 10 10, 20 25, 50 60)
```

Note that point coordinate pairs are separated by commas.

- A Polygon with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A MultiPoint with three Point values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- A MultiLineString with two LineString values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- A MultiPolygon with two Polygon values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5))
```

- A GeometryCollection consisting of two Point values and one LineString:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15,
```

A Backus-Naur grammar that specifies the formal production rules for writing WKT values can be found in the OpenGIS specification document referenced near the beginning of this chapter.

16.3.2. Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation for geometric values is defined by the OpenGIS specification. It is also defined in the ISO *SQL/MM Part 3: Spatial* standard.

WKB is used to exchange geometry data as binary streams represented by BLOB values containing geometric WKB information.

WKB uses one-byte unsigned integers, four-byte unsigned integers, and eight-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to POINT(1 1) consists of this sequence of 21 bytes (each represented here by two hex digits):

```
010100000000000000000000F03F000000000000F03F
```

The sequence may be broken down into these components:

```
Byte order : 01
WKB type   : 01000000
X          : 00000000000000F03F
Y          : 00000000000000F03F
```

Component representation is as follows:

- The byte order may be either 0 or 1 to indicate little-endian or big-endian

storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.

- The WKB type is a code that indicates the geometry type. Values from 1 through 7 indicate Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection.
- A Point value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values are represented by more complex data structures, as detailed in the OpenGIS specification.

16.4. Creating a Spatially Enabled MySQL Database

This section describes the data types you can use for representing spatial data in MySQL, and the functions available for creating and retrieving spatial values.

16.4.1. MySQL Spatial Data Types

MySQL has data types that correspond to OpenGIS classes. Some of these types hold single geometry values:

- GEOMETRY
- POINT
- LINESTRING
- POLYGON

GEOMETRY can store geometry values of any type. The other single-value types (POINT, LINESTRING, and POLYGON) restrict their values to a particular geometry type.

The other data types hold collections of values:

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

GEOMETRYCOLLECTION can store a collection of objects of any type. The other collection types (MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, and GEOMETRYCOLLECTION) restrict collection members to those having a particular geometry type.

16.4.2. Creating Spatial Values

This section describes how to create spatial values using Well-Known Text and Well-Known Binary functions that are defined in the OpenGIS standard, and using MySQL-specific functions.

16.4.2.1. Creating Geometry Values Using WKT Functions

MySQL provides a number of functions that take as input parameters a Well-Known Text representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromText()` accepts a WKT of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromText(wkt[,srid])`,
`GeometryCollectionFromText(wkt[,srid])`

Constructs a `GEOMETRYCOLLECTION` value using its WKT representation and SRID.

- `GeomFromText(wkt[,srid])`, `GeometryFromText(wkt[,srid])`

Constructs a geometry value of any type using its WKT representation and SRID.

- `LineFromText(wkt[,srid])`, `LineStringFromText(wkt[,srid])`

Constructs a `LINestring` value using its WKT representation and SRID.

- `MLineFromText(wkt[,srid])`, `MultiLineStringFromText(wkt[,srid])`

Constructs a `MULTILINestring` value using its WKT representation and SRID.

- `MPointFromText(wkt[,srid])`, `MultiPointFromText(wkt[,srid])`

Constructs a `MULTIPOINT` value using its WKT representation and SRID.

- `MPolyFromText(wkt[,srid])`, `MultiPolygonFromText(wkt[,srid])`

Constructs a MULTIPOLYGON value using its WKT representation and SRID.

- `PointFromText(wkt[,srid])`

Constructs a POINT value using its WKT representation and SRID.

- `PolyFromText(wkt[,srid]), PolygonFromText(wkt[,srid])`

Constructs a POLYGON value using its WKT representation and SRID.

The OpenGIS specification also defines the following optional functions, which MySQL does not implement. These functions construct Polygon or MultiPolygon values based on the WKT representation of a collection of rings or closed LineString values. These values may intersect.

- `BdMPolyFromText(wkt,srid)`

Constructs a MultiPolygon value from a MultiLineString value in WKT format containing an arbitrary collection of closed LineString values.

- `BdPolyFromText(wkt,srid)`

Constructs a Polygon value from a MultiLineString value in WKT format containing an arbitrary collection of closed LineString values.

16.4.2.2. Creating Geometry Values Using WKB Functions

MySQL provides a number of functions that take as input parameters a BLOB containing a Well-Known Binary representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromWKB()` accepts a WKB of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromWKB(wkb[,srid]), GeometryCollectionFromWKB(wkb[,srid])`

Constructs a GEOMETRYCOLLECTION value using its WKB representation and SRID.

- `GeomFromWKB(wkb[,srid])`, `GeometryFromWKB(wkb[,srid])`

Constructs a geometry value of any type using its WKB representation and SRID.

- `LineFromWKB(wkb[,srid])`, `LineStringFromWKB(wkb[,srid])`

Constructs a `LINESTRING` value using its WKB representation and SRID.

- `MLineFromWKB(wkb[,srid])`, `MultiLineStringFromWKB(wkb[,srid])`

Constructs a `MULTILINESTRING` value using its WKB representation and SRID.

- `MPointFromWKB(wkb[,srid])`, `MultiPointFromWKB(wkb[,srid])`

Constructs a `MULTIPOINT` value using its WKB representation and SRID.

- `MPolyFromWKB(wkb[,srid])`, `MultiPolygonFromWKB(wkb[,srid])`

Constructs a `MULTIPOLYGON` value using its WKB representation and SRID.

- `PointFromWKB(wkb[,srid])`

Constructs a `POINT` value using its WKB representation and SRID.

- `PolyFromWKB(wkb[,srid])`, `PolygonFromWKB(wkb[,srid])`

Constructs a `POLYGON` value using its WKB representation and SRID.

The OpenGIS specification also describes optional functions for constructing `Polygon` or `MultiPolygon` values based on the WKB representation of a collection of rings or closed `LineString` values. These values may intersect. MySQL does not implement these functions:

- `BdMPolyFromWKB(wkb,srid)`

Constructs a `MultiPolygon` value from a `MultiLineString` value in WKB format containing an arbitrary collection of closed `LineString` values.

- `BdPolyFromWKB(wkb,srid)`

Constructs a Polygon value from a MultiLineString value in WKB format containing an arbitrary collection of closed LineString values.

16.4.2.3. Creating Geometry Values Using MySQL-Specific Functions

MySQL provides a set of useful non-standard functions for creating geometry WKB representations. The functions described in this section are MySQL extensions to the OpenGIS specification. The results of these functions are BLOB values containing WKB representations of geometry values with no SRID. The results of these functions can be substituted as the first argument for any function in the `GeomFromWKB()` function family.

- `GeometryCollection(g1,g2,...)`

Constructs a WKB `GeometryCollection`. If any argument is not a well-formed WKB representation of a geometry, the return value is `NULL`.

- `LineString(pt1,pt2,...)`

Constructs a WKB `LineString` value from a number of WKB `Point` arguments. If any argument is not a WKB `Point`, the return value is `NULL`. If the number of `Point` arguments is less than two, the return value is `NULL`.

- `MultiLineString(ls1,ls2,...)`

Constructs a WKB `MultiLineString` value using WKB `LineString` arguments. If any argument is not a WKB `LineString`, the return value is `NULL`.

- `MultiPoint(pt1,pt2,...)`

Constructs a WKB `MultiPoint` value using WKB `Point` arguments. If any argument is not a WKB `Point`, the return value is `NULL`.

- `MultiPolygon(poly1,poly2,...)`

Constructs a WKB `MultiPolygon` value from a set of WKB `Polygon` arguments. If any argument is not a WKB `Polygon`, the return value is `NULL`.

- `Point(x,y)`

Constructs a WKB Point using its coordinates.

- `Polygon(ls1,ls2,...)`

Constructs a WKB Polygon value from a number of WKB LineString arguments. If any argument does not represent the WKB of a LinearRing (that is, not a closed and simple LineString) the return value is NULL.

16.4.3. Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Currently, spatial columns are supported for MyISAM, InnoDB, NDB, BDB, and ARCHIVE tables. (Support for storage engines other than MyISAM was added in MySQL 5.0.16.) See also the annotations about spatial indexes under [Section 16.6.1, “Creating Spatial Indexes”](#).

- Use the `CREATE TABLE` statement to create a table with a spatial column:

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the `ALTER TABLE` statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;  
ALTER TABLE geom DROP pt;
```

16.4.4. Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values into internal geometry format:

- Perform the conversion directly in the `INSERT` statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));
```

```
SET @g = 'POINT(1 1)';
```

```
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Perform the conversion prior to the INSERT:

```
SET @g = GeomFromText('POINT(1 1)');  
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';  
INSERT INTO geom VALUES (GeomFromText(@g));
```

```
SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';  
INSERT INTO geom VALUES (GeomFromText(@g));
```

```
SET @g =  
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';  
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples all use `GeomFromText()` to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';  
INSERT INTO geom VALUES (PointFromText(@g));
```

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';  
INSERT INTO geom VALUES (LineStringFromText(@g));
```

```
SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';  
INSERT INTO geom VALUES (PolygonFromText(@g));
```

```
SET @g =  
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';  
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

Note that if a client application program wants to use WKB representations of geometry values, it is responsible for sending correctly formed WKB in queries to the server. However, there are several ways of satisfying this requirement. For example:

- Inserting a `POINT(1 1)` value with hex literal syntax:

```
mysql> INSERT INTO geom VALUES  
-> (GeomFromWKB(0x010100000000000000000000F03F000000000000F03F
```

- An ODBC application can send a WKB representation, binding it to a

placeholder using an argument of BLOB type:

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See [Section 22.2.3.52](#), [“mysql_real_escape_string\(\)”](#).

16.4.5. Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them into WKT or WKB format.

- Fetching spatial data in internal format:

Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

- Fetching spatial data in WKT format:

The `AsText()` function converts a geometry from internal format into a WKT string.

```
SELECT AsText(g) FROM geom;
```

- Fetching spatial data in WKB format:

The `AsBinary()` function converts a geometry from internal format into a BLOB containing the WKB value.

```
SELECT AsBinary(g) FROM geom;
```

16.5. Analyzing Spatial Information

After populating spatial columns with values, you are ready to query and analyze them. MySQL provides a set of functions to perform various operations on spatial data. These functions can be grouped into four major categories according to the type of operation they perform:

- Functions that convert geometries between various formats
- Functions that provide access to qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

Spatial analysis functions can be used in many contexts, such as:

- Any interactive SQL program, such as **mysql** or MySQL Query Browser
- Application programs written in any language that supports a MySQL client API

16.5.1. Geometry Format Conversion Functions

MySQL supports the following functions for converting geometry values between internal format and either WKT or WKB format:

- `AsBinary(g)`

Converts a value in internal geometry format to its WKB representation and returns the binary result.

```
SELECT AsBinary(g) FROM geom;
```

- `AsText(g)`

Converts a value in internal geometry format to its WKT representation and returns the string result.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+-----+
| AsText(GeomFromText(@g)) |
+-----+
| LINESTRING(1 1,2 2,3 3) |
+-----+
```

- `GeomFromText(wkt[,srid])`

Converts a string value from its WKT representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as `PointFromText()` and `LineFromText()`. See [Section 16.4.2.1, “Creating Geometry Values Using WKT Functions”](#).

- `GeomFromWKB(wkb[,srid])`

Converts a binary value from its WKB representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as `PointFromWKB()` and `LineFromWKB()`. See [Section 16.4.2.2, “Creating Geometry Values Using WKB Functions”](#).

16.5.2. Geometry Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return `NULL` if the argument is of an incorrect geometry type. For example, `Area()` returns `NULL` if the object type is neither `Polygon` nor `MultiPolygon`.

16.5.2.1. General Geometry Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- `Dimension(g)`

Returns the inherent dimension of the geometry value *g*. The result can be `-1`, `0`, `1`, or `2`. The meaning of these values is given in [Section 16.2.2, “Class Geometry”](#).

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| 1 |
+-----+
```

- Envelope(g)

Returns the Minimum Bounding Rectangle (MBR) for the geometry value *g*. The result is returned as a Polygon value.

The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)'))
+-----+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)')) |
+-----+
| POLYGON((1 1,2 1,2 2,1 2,1 1)) |
+-----+
```

- GeometryType(g)

Returns as a string the name of the geometry type of which the geometry instance *g* is a member. The name corresponds to one of the instantiable Geometry subclasses.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----+
| POINT |
+-----+
```

- SRID(g)

Returns an integer indicating the Spatial Reference System ID for the geometry value *g*.

In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
```

```

+-----+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----+
|                                                    101 |
+-----+

```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- `Boundary(g)`

Returns a geometry that is the closure of the combinatorial boundary of the geometry value *g*.

- `IsEmpty(g)`

Returns 1 if the geometry value *g* is the empty geometry, 0 if it is not empty, and -1 if the argument is NULL. If the geometry is empty, it represents the empty point set.

- `IsSimple(g)`

Currently, this function is a placeholder and should not be used. If implemented, its behavior will be as described in the next paragraph.

Returns 1 if the geometry value *g* has no anomalous geometric points, such as self-intersection or self-tangency. `IsSimple()` returns 0 if the argument is not simple, and -1 if it is NULL.

The description of each instantiable geometric class given earlier in the chapter includes the specific conditions that cause an instance of that class to be classified as not simple. (See [Section 16.2.1, “The Geometry Class Hierarchy”](#).)

16.5.2.2. Point Functions

A `Point` consists of X and Y coordinates, which may be obtained using the following functions:

- `X(p)`

Returns the X-coordinate value for the point p as a double-precision number.

```
mysql> SET @pt = 'Point(56.7 53.34)';
mysql> SELECT X(GeomFromText(@pt));
+-----+
| X(GeomFromText(@pt)) |
+-----+
|                56.7 |
+-----+
```

- $Y(p)$

Returns the Y-coordinate value for the point p as a double-precision number.

```
mysql> SET @pt = 'Point(56.7 53.34)';
mysql> SELECT Y(GeomFromText(@pt));
+-----+
| Y(GeomFromText(@pt)) |
+-----+
|                53.34 |
+-----+
```

16.5.2.3. LineString Functions

A `LineString` consists of `Point` values. You can extract particular points of a `LineString`, count the number of points that it contains, or obtain its length.

- `EndPoint(ls)`

Returns the `Point` that is the endpoint of the `LineString` value ls .

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-----+
| AsText(EndPoint(GeomFromText(@ls))) |
+-----+
| POINT(3 3) |
+-----+
```

- `GLength(ls)`

Returns as a double-precision number the length of the `LineString` value

ls in its associated spatial reference.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+-----+
| GLength(GeomFromText(@ls)) |
+-----+
|                2.8284271247462 |
+-----+
```

GLength() is a non-standard name. It corresponds to the OpenGIS Length() function.

- NumPoints(*ls*)

Returns the number of Point objects in the LineString value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+-----+
| NumPoints(GeomFromText(@ls)) |
+-----+
|                                3 |
+-----+
```

- PointN(*ls*,*N*)

Returns the *N*-th Point in the LineString value *ls*. Points are numbered beginning with 1.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
+-----+
| AsText(PointN(GeomFromText(@ls),2)) |
+-----+
| POINT(2 2) |
+-----+
```

- StartPoint(*ls*)

Returns the Point that is the start point of the LineString value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+-----+
| AsText(StartPoint(GeomFromText(@ls))) |
```

```

+-----+
| POINT(1 1) |
+-----+

```

The OpenGIS specification also defines the following function, which MySQL does not implement:

- `IsRing(ls)`

Returns 1 if the `LineString` value *ls* is closed (that is, its `StartPoint()` and `EndPoint()` values are the same) and is simple (does not pass through the same point more than once). Returns 0 if *ls* is not a ring, and -1 if it is `NULL`.

16.5.2.4. MultiLineString Functions

- `GLength(mls)`

Returns as a double-precision number the length of the `MultiLineString` value *mls*. The length of *mls* is equal to the sum of the lengths of its elements.

```

mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+-----+
| GLength(GeomFromText(@mls)) |
+-----+
|                4.2426406871193 |
+-----+

```

`GLength()` is a non-standard name. It corresponds to the OpenGIS `Length()` function.

- `IsClosed(mls)`

Returns 1 if the `MultiLineString` value *mls* is closed (that is, the `StartPoint()` and `EndPoint()` values are the same for each `LineString` in *mls*). Returns 0 if *mls* is not closed, and -1 if it is `NULL`.

```

mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+-----+
| IsClosed(GeomFromText(@mls)) |

```

```

+-----+
|                                     0 |
+-----+

```

16.5.2.5. Polygon Functions

- Area(poly)

Returns as a double-precision number the area of the Polygon value *poly*, as measured in its spatial reference system.

```

mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))
mysql> SELECT Area(GeomFromText(@poly));
+-----+
| Area(GeomFromText(@poly)) |
+-----+
|                             4 |
+-----+

```

- ExteriorRing(poly)

Returns the exterior ring of the Polygon value *poly* as a LineString.

```

mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-----+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-----+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)           |
+-----+

```

- InteriorRingN(poly,N)

Returns the *N*-th interior ring for the Polygon value *poly* as a LineString. Rings are numbered beginning with 1.

```

mysql> SET @poly =
-> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+-----+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+-----+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)           |
+-----+

```

- NumInteriorRings(poly)

Returns the number of interior rings in the Polygon value *poly*.

```
mysql> SET @poly =
      -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+-----+
| NumInteriorRings(GeomFromText(@poly)) |
+-----+
|                                     1 |
+-----+
```

16.5.2.6. MultiPolygon Functions

- Area(mpoly)

Returns as a double-precision number the area of the MultiPolygon value *mpoly*, as measured in its spatial reference system.

```
mysql> SET @mpoly =
      -> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1
mysql> SELECT Area(GeomFromText(@mpoly));
+-----+
| Area(GeomFromText(@mpoly)) |
+-----+
|                             8 |
+-----+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- Centroid(mpoly)

Returns the mathematical centroid for the MultiPolygon value *mpoly* as a Point. The result is not guaranteed to be on the MultiPolygon.

- PointOnSurface(mpoly)

Returns a Point value that is guaranteed to be on the MultiPolygon value *mpoly*.

16.5.2.7. GeometryCollection Functions

- `GeometryN(gc,N)`

Returns the *N*-th geometry in the `GeometryCollection` value *gc*. Geometries are numbered beginning with 1.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2,
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+-----+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+-----+
| POINT(1 1) |
+-----+
```

- `NumGeometries(gc)`

Returns the number of geometries in the `GeometryCollection` value *gc*.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2,
mysql> SELECT NumGeometries(GeomFromText(@gc));
+-----+
| NumGeometries(GeomFromText(@gc)) |
+-----+
| 2 |
+-----+
```

16.5.3. Functions That Create New Geometries from Existing Ones

16.5.3.1. Geometry Functions That Produce New Geometries

[Section 16.5.2, “Geometry Functions”](#), discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- `Envelope(g)`
- `StartPoint(ls)`
- `EndPoint(ls)`
- `PointN(ls,N)`

- ExteriorRing(poly)
- InteriorRingN(poly,N)
- GeometryN(gc,N)

16.5.3.2. Spatial Operators

OpenGIS proposes a number of other functions that can produce geometries. They are designed to implement spatial operators.

These functions are not implemented in MySQL. They may appear in future releases.

- Buffer(g,d)

Returns a geometry that represents all points whose distance from the geometry value g is less than or equal to a distance of d .

- ConvexHull(g)

Returns a geometry that represents the convex hull of the geometry value g .

- Difference($g1,g2$)

Returns a geometry that represents the point set difference of the geometry value $g1$ with $g2$.

- Intersection($g1,g2$)

Returns a geometry that represents the point set intersection of the geometry values $g1$ with $g2$.

- SymDifference($g1,g2$)

Returns a geometry that represents the point set symmetric difference of the geometry value $g1$ with $g2$.

- Union($g1,g2$)

Returns a geometry that represents the point set union of the geometry values *g1* and *g2*.

16.5.4. Functions for Testing Spatial Relations Between Geometric Objects

The functions described in these sections take two geometries as input parameters and return a qualitative or quantitative relation between them.

16.5.5. Relations on Geometry Minimal Bounding Rectangles (MBRs)

MySQL provides several functions that test relations between minimal bounding rectangles of two geometries *g1* and *g2*. The return values 1 and 0 indicate true and false, respectively.

- `MBRContains(g1,g2)`

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* contains the Minimum Bounding Rectangle of *g2*.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
+-----+-----+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

- `MBRDisjoint(g1,g2)`

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

- `MBREqual(g1,g2)`

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are the same.

- `MBRIntersects(g1,g2)`

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* intersect.

- `MBROverlaps(g1,g2)`

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* overlap.

- `MBRTouches(g1,g2)`

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* touch.

- `MBRWithin(g1,g2)`

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* is within the Minimum Bounding Rectangle of *g2*.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-----+-----+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

16.5.6. Functions That Test Spatial Relationships Between Geometries

The OpenGIS specification defines the following functions. They test the relationship between two geometry values *g1* and *g2*.

Currently, MySQL does not implement these functions according to the specification. Those that are implemented return the same result as the corresponding MBR-based functions. This includes functions in the following list other than `Distance()` and `Related()`.

These functions may be implemented in future releases with full support for spatial analysis, not just MBR-based support.

The return values 1 and 0 indicate true and false, respectively.

- `Contains(g1,g2)`

Returns 1 or 0 to indicate whether *g1* completely contains *g2*.

- `Crosses(g1,g2)`

Returns 1 if *g1* spatially crosses *g2*. Returns NULL if *g1* is a Polygon or a MultiPolygon, or if *g2* is a Point or a MultiPoint. Otherwise, returns 0.

The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

- The two geometries intersect
- Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
- Their intersection is not equal to either of the two given geometries

- `Disjoint(g1,g2)`

Returns 1 or 0 to indicate whether *g1* is spatially disjoint from (does not intersect) *g2*.

- `Distance(g1,g2)`

Returns as a double-precision number the shortest distance between any two points in the two geometries.

- `Equals(g1,g2)`

Returns 1 or 0 to indicate whether *g1* is spatially equal to *g2*.

- `Intersects(g1,g2)`

Returns 1 or 0 to indicate whether *g1* spatially intersects *g2*.

- `Overlaps(g1,g2)`

Returns 1 or 0 to indicate whether *g1* spatially overlaps *g2*. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `Related(g1,g2,pattern_matrix)`

Returns 1 or 0 to indicate whether the spatial relationship specified by *pattern_matrix* exists between *g1* and *g2*. Returns -1 if the arguments are NULL. The pattern matrix is a string. Its specification will be noted here if this function is implemented.

- `Touches(g1,g2)`

Returns 1 or 0 to indicate whether *g1* spatially touches *g2*. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `Within(g1,g2)`

Returns 1 or 0 to indicate whether *g1* is spatially within *g2*.

16.6. Optimizing Spatial Analysis

Search operations in non-spatial databases can be optimized using indexes. This is true for spatial databases as well. With the help of a great variety of multi-dimensional indexing methods that have previously been designed, it is possible to optimize spatial searches. The most typical of these are:

- Point queries that search for all objects that contain a given point
- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** to index spatial columns. A spatial index is built using the MBR of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. Beginning with MySQL 5.0.16, you must declare a prefix for any (non-spatial) index on a spatial column except for POINT columns.

16.6.1. Creating Spatial Indexes

MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but extended with the SPATIAL keyword. Currently, spatial columns that are indexed must be declared NOT NULL. The following examples demonstrate how to create spatial indexes:

- With CREATE TABLE:

```
CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

- With ALTER TABLE:

```
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- With CREATE INDEX:

```
CREATE SPATIAL INDEX sp_index ON geom (g);
```

For MyISAM tables, `SPATIAL INDEX` creates an R-tree index. For other storage engines that support spatial indexing, `SPATIAL INDEX` creates a B-tree index. A B-tree index on spatial values will be useful for exact-value lookups, but not for range scans.

To drop spatial indexes, use `ALTER TABLE` or `DROP INDEX`:

- With `ALTER TABLE`:

```
ALTER TABLE geom DROP INDEX g;
```

- With `DROP INDEX`:

```
DROP INDEX sp_index ON geom;
```

Example: Suppose that a table `geom` contains more than 32,000 geometries, which are stored in the column `g` of type `GEOMETRY`. The table also has an `AUTO_INCREMENT` column `fid` for storing object ID values.

```
mysql> DESCRIBE geom;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| fid   | int(11)   |      | PRI | NULL    | auto_increment |
| g     | geometry  |      |     |         |                 |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM geom;
```

```
+-----+
| count(*) |
+-----+
|    32376 |
+-----+
1 row in set (0.00 sec)
```

To add a spatial index on the column `g`, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
```

```
Query OK, 32376 rows affected (4.05 sec)
Records: 32376 Duplicates: 0 Warnings: 0
```

16.6.2. Using a Spatial Index

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as `MBRContains()` or `MBRWithin()` in the `WHERE` clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,31000 15000,31000 16000,30000 16000,300
mysql> SELECT fid,AsText(g) FROM geom WHERE
-> MBRContains(GeomFromText(@poly),g);
```

fid	AsText(g)
21	LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30
22	LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ..
23	LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ..
24	LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ..
25	LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ..
26	LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ..
249	LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ..
1	LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ..
2	LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ..
3	LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ..
4	LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ..
5	LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ..
6	LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ..
7	LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ..
10	LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ..
11	LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ..
13	LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ..
154	LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ..
155	LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ..
157	LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ..

```
20 rows in set (0.00 sec)
```

Use `EXPLAIN` to check the way this query is executed:

```
mysql> SET @poly =
-> 'Polygon((30000 15000,31000 15000,31000 16000,30000 16000,300
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
-> MBRContains(GeomFromText(@poly),g)\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: geom
         type: range
possible_keys: g
          key: g
```

```

    key_len: 32
      ref: NULL
      rows: 50
    Extra: Using where
1 row in set (0.00 sec)

```

Check what would happen without a spatial index:

```

mysql> SET @poly =
      -> 'Polygon((30000 15000,31000 15000,31000 16000,30000 16000,300
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
      -> MBRContains(GeomFromText(@poly),g)\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: geom
      type: ALL
possible_keys: NULL
      key: NULL
    key_len: NULL
      ref: NULL
      rows: 32376
    Extra: Using where
1 row in set (0.00 sec)

```

Executing the SELECT statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```

mysql> SET @poly =
      -> 'Polygon((30000 15000,31000 15000,31000 16000,30000 16000,300
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
      -> MBRContains(GeomFromText(@poly),g);
+-----+-----+-----+-----+-----+-----+-----+-----+
| fid | AsText(g)
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | LINestring(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ..
| 2 | LINestring(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ..
| 3 | LINestring(30179 15114.4,30176.6 15129.4,30167 15128,3016 ..
| 4 | LINestring(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ..
| 5 | LINestring(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ..
| 6 | LINestring(30244 15087,30229 15086.2,30229.4 15076.4,3024 ..
| 7 | LINestring(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ..
| 10 | LINestring(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ..
| 11 | LINestring(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ..
| 13 | LINestring(30105 15065.8,30108.4 15050.8,30118 15053,3011 ..
| 21 | LINestring(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ..
| 22 | LINestring(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ..
| 23 | LINestring(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ..

```

```

| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ..
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ..
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ..
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ..
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ..
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ..
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ..
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
20 rows in set (0.46 sec)

```

In future releases, spatial indexes may also be used for optimizing other functions. See [Section 16.5.4, “Functions for Testing Spatial Relations Between Geometric Objects”](#).

16.7. MySQL Conformance and Compatibility

MySQL does not yet implement the following GIS features:

- Additional Metadata Views

OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function `Length()` on `LineString` and `MultiLineString` currently should be called in MySQL as `GLength()`

The problem is that there is an existing SQL function `Length()` that calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context. We need either to solve this somehow, or decide on another function name.

Chapter 17. Stored Procedures and Functions

Table of Contents

[17.1. Stored Routines and the Grant Tables](#)

[17.2. Stored Routine Syntax](#)

[17.2.1. CREATE PROCEDURE and CREATE FUNCTION Syntax](#)

[17.2.2. ALTER PROCEDURE and ALTER FUNCTION Syntax](#)

[17.2.3. DROP PROCEDURE and DROP FUNCTION Syntax](#)

[17.2.4. CALL Statement Syntax](#)

[17.2.5. BEGIN . . . END Compound Statement Syntax](#)

[17.2.6. DECLARE Statement Syntax](#)

[17.2.7. Variables in Stored Routines](#)

[17.2.8. Conditions and Handlers](#)

[17.2.9. Cursors](#)

[17.2.10. Flow Control Constructs](#)

[17.3. Stored Procedures, Functions, Triggers, and Replication: Frequently Asked Questions](#)

[17.4. Binary Logging of Stored Routines and Triggers](#)

Stored routines (procedures and functions) are supported in MySQL 5.0. A stored procedure is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored procedure instead.

Some situations where stored routines can be particularly useful:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures and functions for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Stored routines can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does

increase the load on the database server because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored routines also allow you to have libraries of functions in the database server. This is a feature shared by modern application languages that allow such design internally (for example, by using classes). Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2.

The MySQL implementation of stored routines is still in progress. All syntax described in this chapter is supported and any limitations and extensions are documented where appropriate. Further discussion of restrictions on use of stored routines is given in [Section I.1, “Restrictions on Stored Routines and Triggers”](#).

Binary logging for stored routines takes place as described in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

Recursive stored procedures are disabled by default, but can be enabled on the server by setting the `max_sp_recursion_depth` server system variable to a nonzero value. See [Section 5.2.2, “Server System Variables”](#), for more information.

Stored functions cannot be recursive. See [Section I.1, “Restrictions on Stored Routines and Triggers”](#).

17.1. Stored Routines and the Grant Tables

Stored routines require the `proc` table in the `mysql` database. This table is created during the MySQL 5.0 installation procedure. If you are upgrading to MySQL 5.0 from an earlier version, be sure to update your grant tables to make sure that the `proc` table exists. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

The server manipulates the `mysql.proc` table in response to statements that create, alter, or drop stored routines. It is not supported that the server will notice manual manipulation of this table.

Beginning with MySQL 5.0.3, the grant system takes stored routines into account as follows:

- The `CREATE ROUTINE` privilege is needed to create stored routines.
- The `ALTER ROUTINE` privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine.
- The `EXECUTE` privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a routine. Also, the default `SQL SECURITY` characteristic for a routine is `DEFINER`, which enables users who have access to the database with which the routine is associated to execute the routine.

17.2. Stored Routine Syntax

A stored routine is either a procedure or a function. Stored routines are created with `CREATE PROCEDURE` and `CREATE FUNCTION` statements. A procedure is invoked using a `CALL` statement, and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. Stored routines may call other stored routines.

As of MySQL 5.0.1, a stored procedure or function is associated with a particular database. This has several implications:

- When the routine is invoked, an implicit `USE db_name` is performed (and undone when the routine terminates). `USE` statements within stored routines are disallowed.
- You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure `p` or function `f` that is associated with the `test` database, you can say `CALL test.p()` or `test.f()`.
- When a database is dropped, all stored routines associated with it are dropped as well.

(In MySQL 5.0.0, stored routines are global and not associated with a database. They inherit the default database from the caller. If a `USE db_name` is executed within the routine, the original default database is restored upon routine exit.)

MySQL supports the very useful extension that allows the use of regular `SELECT` statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple `SELECT` statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1. The client should also specify the `CLIENT_MULTI_STATEMENTS` option when it connects. For C programs, this can be done with the `mysql_real_connect()` C API function (see [Section 22.2.3.51, “mysql_real_connect\(\)”](#)).

The following sections describe the syntax used to create, alter, drop, and invoke stored procedures and functions.

17.2.1. CREATE PROCEDURE and CREATE FUNCTION Syntax

```
CREATE
  [DEFINER = { user | CURRENT_USER }]
  PROCEDURE sp_name ([proc_parameter[, ...]])
  [characteristic ...] routine_body
```

```
CREATE
  [DEFINER = { user | CURRENT_USER }]
  FUNCTION sp_name ([func_parameter[, ...]])
  RETURNS type
  [characteristic ...] routine_body
```

```
proc_parameter:
  [ IN | OUT | INOUT ] param_name type
```

```
func_parameter:
  param_name type
```

```
type:
  Any valid MySQL data type
```

```
characteristic:
  LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

```
routine_body:
  Valid SQL procedure statement
```

These statements create stored routines. As of MySQL 5.0.3, to use them, it is necessary to have the CREATE ROUTINE privilege. If binary logging is enabled, these statements might also require the SUPER privilege, as described in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#). MySQL automatically grants the ALTER ROUTINE and EXECUTE privileges to the routine creator.

By default, the routine is associated with the default database. To associate the routine explicitly with a given database, specify the name as *db_name.sp_name* when you create it.

If the routine name is the same as the name of a built-in SQL function, you must use a space between the name and the following parenthesis when defining the routine, or a syntax error occurs. This is also true when you invoke the routine later. For this reason, we suggest that it is better to avoid re-using the names of existing SQL functions for your own stored routines.

The `IGNORE_SPACE` SQL mode applies to built-in functions, not to stored routines. It is always allowable to have spaces after a routine name, regardless of whether `IGNORE_SPACE` is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of `()` should be used.

Each parameter can be declared to use any valid data type, except that the `COLLATE` attribute cannot be used.

Each parameter is an `IN` parameter by default. To specify otherwise for a parameter, use the keyword `OUT` or `INOUT` before the parameter name.

Note: Specifying a parameter as `IN`, `OUT`, or `INOUT` is valid only for a `PROCEDURE`. (`FUNCTION` parameters are always regarded as `IN` parameters.)

An `IN` parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An `OUT` parameter passes a value from the procedure back to the caller. Its initial value is `NULL` within the procedure, and its value is visible to the caller when the procedure returns. An `INOUT` parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each `OUT` or `INOUT` parameter, pass a user-defined variable so that you can obtain its value when the procedure returns. (For an example, see [Section 17.2.4, “CALL Statement Syntax”](#).) If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.

The `RETURNS` clause may be specified only for a `FUNCTION`, for which it is mandatory. It indicates the return type of the function, and the function body must contain a `RETURN value` statement.

The *routine_body* consists of a valid SQL procedure statement. This can be a simple statement such as SELECT or INSERT, or it can be a compound statement written using BEGIN and END. Compound statement syntax is described in [Section 17.2.5, “BEGIN . . . END Compound Statement Syntax”](#). Compound statements can contain declarations, loops, and other control structure statements. The syntax for these statements is described later in this chapter. See, for example, [Section 17.2.6, “DECLARE Statement Syntax”](#), and [Section 17.2.10, “Flow Control Constructs”](#). Some statements are not allowed in stored routines; see [Section I.1, “Restrictions on Stored Routines and Triggers”](#).

The CREATE FUNCTION statement was used in earlier versions of MySQL to support UDFs (user-defined functions). See [Section 24.2, “Adding New Functions to MySQL”](#). UDFs continue to be supported, even with the existence of stored functions. A UDF can be regarded as an external stored function. However, do note that stored functions share their namespace with UDFs.

A procedure or function is considered “deterministic” if it always produces the same result for the same input parameters, and “not deterministic” otherwise. If neither DETERMINISTIC nor NOT DETERMINISTIC is given in the routine definition, the default is NOT DETERMINISTIC.

A routine that contains the NOW() function (or its synonyms) or RAND() is non-deterministic, but it might still be replication-safe. For NOW(), the binary log includes the timestamp and replicates correctly. RAND() also replicates correctly as long as it is invoked only once within a routine. (You can consider the routine execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

Currently, the DETERMINISTIC characteristic is accepted, but not yet used by the optimizer. However, if binary logging is enabled, this characteristic affects which routine definitions MySQL accepts. See [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

Several characteristics provide information about the nature of data use by the routine. CONTAINS SQL indicates that the routine does not contain statements that read or write data. NO SQL indicates that the routine contains no SQL statements. READS SQL DATA indicates that the routine contains statements that read data, but not statements that write data. MODIFIES SQL DATA indicates that the routine contains statements that may write data. CONTAINS SQL is the default if none of

these characteristics is given explicitly. These characteristics are advisory only. The server does not use them to constrain what kinds of statements a routine will be allowed to execute.

The SQL SECURITY characteristic can be used to specify whether the routine should be executed using the permissions of the user who creates the routine or the user who invokes it. The default value is DEFINER. This feature is new in SQL:2003. The creator or invoker must have permission to access the database with which the routine is associated. As of MySQL 5.0.3, it is necessary to have the EXECUTE privilege to be able to execute the routine. The user that must have this privilege is either the definer or invoker, depending on how the SQL SECURITY characteristic is set.

The optional DEFINER clause specifies the MySQL account to be used when checking access privileges at routine execution time for routines that have the SQL SECURITY DEFINER characteristic. The DEFINER clause was added in MySQL 5.0.20.

If a *user* value is given, it should be a MySQL account in 'user_name'@'host_name' format (the same format used in the GRANT statement). The *user_name* and *host_name* values both are required. CURRENT_USER also can be given as CURRENT_USER(). The default DEFINER value is the user who executes the CREATE PROCEDURE or CREATE FUNCTION or statement. (This is the same as DEFINER = CURRENT_USER.)

If you specify the DEFINER clause, you cannot set the value to any account but your own unless you have the SUPER privilege. These rules determine the legal DEFINER user values:

- If you do not have the SUPER privilege, the only legal *user* value is your own account, either specified literally or by using CURRENT_USER. You cannot set the definer to some other account.
- If you have the SUPER privilege, you can specify any syntactically legal account name. If the account does not actually exist, a warning is generated.

Although it is possible to create routines with a non-existent DEFINER value, an error occurs if the routine executes with definer privileges but the definer does not exist at execution time.

MySQL stores the `sql_mode` system variable setting that is in effect at the time a routine is created, and always executes the routine with this setting in force.

When the routine is invoked, an implicit `USE db_name` is performed (and undone when the routine terminates). `USE` statements within stored routines are disallowed.

As of MySQL 5.0.18, the server uses the data type of a routine parameter or function return value as follows. These rules also apply to local routine variables created with the `DECLARE` statement ([Section 17.2.7.1, “DECLARE Local Variables”](#)).

- Assignments are checked for data type mismatches and overflow. Conversion and overflow problems result in warnings, or errors in strict mode.
- For character data types, if there is a `CHARACTER SET` clause in the declaration, the specified character set and its default collation are used. If there is no such clause, the database character set and collation are used. (These are given by the values of the `character_set_database` and `collation_database` system variables.)
- Only scalar values can be assigned to parameters or variables. For example, a statement such as `SET x = (SELECT 1, 2)` is invalid.

Before MySQL 5.0.18, parameters, return values, and local variables are treated as items in expressions, and are subject to automatic (silent) conversion and truncation. Stored functions ignore the `sql_mode` setting.

The `COMMENT` clause is a MySQL extension, and may be used to describe the stored routine. This information is displayed by the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements.

MySQL allows routines to contain DDL statements, such as `CREATE` and `DROP`. MySQL also allows stored procedures (but not stored functions) to contain SQL transaction statements such as `COMMIT`. Stored functions may not contain statements that do explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to allow them.

Stored routines cannot use `LOAD DATA INFILE`.

Statements that return a result set cannot be used within a stored function. This includes `SELECT` statements that do not use `INTO` to fetch column values into variables, `SHOW` statements, and other statements such as `EXPLAIN`. For statements that can be determined at function definition time to return a result set, a Not allowed to return a result set from a function error occurs (`ER_SP_NO_RESET_IN_FUNC`). For statements that can be determined only at runtime to return a result set, a `PROCEDURE %s can't return a result set in the given context` error occurs (`ER_SP_BADSELECT`).

Note: Before MySQL 5.0.10, stored functions created with `CREATE FUNCTION` must not contain references to tables, with limited exceptions. They may include some `SET` statements that contain table references, for example `SET a := (SELECT MAX(id) FROM t)`, and `SELECT` statements that fetch values directly into variables, for example `SELECT i INTO var1 FROM t`.

The following is an example of a simple stored procedure that uses an `OUT` parameter. The example uses the `mysql` client `delimiter` command to change the statement delimiter from `;` to `//` while the procedure is being defined. This allows the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself.

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
-> BEGIN
->   SELECT COUNT(*) INTO param1 FROM t;
-> END;
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+-----+
| @a    |
+-----+
| 3     |
+-----+
1 row in set (0.00 sec)
```

When using the `delimiter` command, you should avoid the use of the backslash (`\`) character because that is the escape character for MySQL.

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
      -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world!  |
+-----+
1 row in set (0.00 sec)
```

A stored function returns a value of the data type specified in its `RETURNS` clause. If the `RETURN` statement returns a value of a different type, the value is coerced to the proper type. For example, if a function returns an `ENUM` or `SET` value, but the `RETURN` statement returns an integer, the value returned from the function is the string for the corresponding `ENUM` member or set of `SET` members.

17.2.2. ALTER PROCEDURE and ALTER FUNCTION Syntax

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]
```

characteristic:

```
    { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
    | SQL SECURITY { DEFINER | INVOKER }
    | COMMENT 'string'
```

This statement can be used to change the characteristics of a stored procedure or function. As of MySQL 5.0.3, you must have the `ALTER ROUTINE` privilege for the routine. (That privilege is granted automatically to the routine creator.) If binary logging is enabled, this statement might also require the `SUPER` privilege, as described in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

More than one change may be specified in an `ALTER PROCEDURE` or `ALTER FUNCTION` statement.

17.2.3. DROP PROCEDURE and DROP FUNCTION Syntax

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

This statement is used to drop a stored procedure or function. That is, the specified routine is removed from the server. As of MySQL 5.0.3, you must have the ALTER ROUTINE privilege for the routine. (That privilege is granted automatically to the routine creator.)

The IF EXISTS clause is a MySQL extension. It prevents an error from occurring if the procedure or function does not exist. A warning is produced that can be viewed with SHOW WARNINGS.

17.2.4. CALL Statement Syntax

```
CALL sp_name([parameter[, ...]])
```

The CALL statement invokes a procedure that was defined previously with CREATE PROCEDURE.

CALL can pass back values to its caller using parameters that are declared as OUT or INOUT parameters. It also “returns” the number of rows affected, which a client program can obtain at the SQL level by calling the ROW_COUNT() function and from C by calling the mysql_affected_rows() C API function.

To get back a value from a procedure using an OUT or INOUT parameter, pass the parameter by means of a user variable, and then check the value of the variable after the procedure returns. (If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an IN or INOUT parameter.) For an INOUT parameter, initialize its value before passing it to the procedure. The following procedure has an OUT parameter that the procedure sets to the current server version, and an INOUT value that the procedure increments by one from its current value:

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;
```

Before calling the procedure, initialize the variable to be passed as the INOUT parameter. After calling the procedure, the values of the two variables will have been set or modified:

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+-----+-----+
| @version | @increment |
+-----+-----+
| 5.0.25-log | 11          |
+-----+-----+
```

If you write C programs that execute stored procedures with the CALL SQL statement, you *must* set the CLIENT_MULTI_RESULTS flag when you call `mysql_real_connect()`, either explicitly, or implicitly by setting CLIENT_MULTI_STATEMENTS. This is because each CALL returns a result to indicate the call status, in addition to any results sets that might be returned by statements executed within the procedure. To process the result of a CALL statement, use a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 22.2.9, “C API Handling of Multiple Statement Execution”](#).

17.2.5. BEGIN ... END Compound Statement Syntax

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

BEGIN ... END syntax is used for writing compound statements, which can appear within stored routines and triggers. A compound statement can contain multiple statements, enclosed by the BEGIN and END keywords. *statement_list* represents a list of one or more statements. Each statement within *statement_list* must be terminated by a semicolon (;) statement delimiter. Note that *statement_list* is optional, which means that the empty compound statement (BEGIN END) is legal.

Use of multiple statements requires that a client is able to send statement strings containing the ; statement delimiter. This is handled in the **mysql** command-line client with the `delimiter` command. Changing the ; end-of-statement delimiter (for example, to //) allows ; to be used in a routine body. For an example, see

[Section 17.2.1, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).

A compound statement can be labeled. *end_label* cannot be given unless *begin_label* also is present. If both are present, they must be the same.

The optional [NOT] ATOMIC clause is not yet supported. This means that no transactional savepoint is set at the start of the instruction block and the BEGIN clause used in this context has no effect on the current transaction.

17.2.6. DECLARE Statement Syntax

The DECLARE statement is used to define various items local to a routine:

- Local variables. See [Section 17.2.7, “Variables in Stored Routines”](#).
- Conditions and handlers. See [Section 17.2.8, “Conditions and Handlers”](#).
- Cursors. See [Section 17.2.9, “Cursors”](#).

The SIGNAL and RESIGNAL statements are not currently supported.

DECLARE is allowed only inside a BEGIN ... END compound statement and must be at its start, before any other statements.

Declarations must follow a certain order. Cursors must be declared before declaring handlers, and variables and conditions must be declared before declaring either cursors or handlers.

17.2.7. Variables in Stored Routines

You may declare and use variables within a routine.

17.2.7.1. DECLARE Local Variables

```
DECLARE var_name[, ...] type [DEFAULT value]
```

This statement is used to declare local variables. To provide a default value for the variable, include a DEFAULT clause. The value can be specified as an expression; it need not be a constant. If the DEFAULT clause is missing, the initial

value is NULL.

Local variables are treated like routine parameters with respect to data type and overflow checking. See [Section 17.2.1, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#).

The scope of a local variable is within the BEGIN ... END block where it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

17.2.7.2. Variable SET Statement

```
SET var_name = expr [, var_name = expr] ...
```

The SET statement in stored routines is an extended version of the general SET statement. Referenced variables may be ones declared inside a routine, or global system variables.

The SET statement in stored routines is implemented as part of the pre-existing SET syntax. This allows an extended syntax of SET a=x, b=y, ... where different variable types (locally declared variables and global and session server variables) can be mixed. This also allows combinations of local variables and some options that make sense only for system variables; in that case, the options are recognized but ignored.

17.2.7.3. SELECT ... INTO Statement

```
SELECT col_name[,...] INTO var_name[,...] table_expr
```

This SELECT syntax stores selected columns directly into variables. Therefore, only a single row may be retrieved.

```
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;
```

User variable names are not case sensitive. See [Section 9.3, “User-Defined Variables”](#).

Important: SQL variable names should not be the same as column names. If an SQL statement, such as a SELECT ... INTO statement, contains a reference to a column and a declared local variable with the same name, MySQL currently

interprets the reference as the name of a variable. For example, in the following statement, `xname` is interpreted as a reference to the `xname` *variable* rather than the `xname` *column*:

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname,id INTO newname,xid
  FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

When this procedure is called, the `newname` variable returns the value 'bob' regardless of the value of the `table1.xname` column.

See also [Section I.1, “Restrictions on Stored Routines and Triggers”](#).

17.2.8. Conditions and Handlers

Certain conditions may require specific handling. These conditions can relate to errors, as well as to general flow control inside a routine.

17.2.8.1. DECLARE Conditions

```
DECLARE condition_name CONDITION FOR condition_value
```

```
condition_value:
  SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code
```

This statement specifies conditions that need specific handling. It associates a name with a specified error condition. The name can subsequently be used in a `DECLARE HANDLER` statement. See [Section 17.2.8.2, “DECLARE Handlers”](#).

A *condition_value* can be an SQLSTATE value or a MySQL error code.

17.2.8.2. DECLARE Handlers

```
DECLARE handler_type HANDLER FOR condition_value[,...] statement
```

```

handler_type:
    CONTINUE
    | EXIT
    | UNDO

condition_value:
    SQLSTATE [VALUE] sqlstate_value
    | condition_name
    | SQLWARNING
    | NOT FOUND
    | SQLEXCEPTION
    | mysql_error_code

```

The `DECLARE . . . HANDLER` statement specifies handlers that each may deal with one or more conditions. If one of these conditions occurs, the specified *statement* is executed. *statement* can be a simple statement (for example, `SET var_name = value`), or it can be a compound statement written using `BEGIN` and `END` (see [Section 17.2.5, “BEGIN . . . END Compound Statement Syntax”](#)).

For a `CONTINUE` handler, execution of the current routine continues after execution of the handler statement. For an `EXIT` handler, execution terminates for the `BEGIN . . . END` compound statement in which the handler is declared. (This is true even if the condition occurs in an inner block.) The `UNDO` handler type statement is not yet supported.

If a condition occurs for which no handler has been declared, the default action is `EXIT`.

A *condition_value* can be any of the following values:

- An `SQLSTATE` value or a MySQL error code.
- A condition name previously specified with `DECLARE . . . CONDITION`. See [Section 17.2.8.1, “DECLARE Conditions”](#).
- `SQLWARNING` is shorthand for all `SQLSTATE` codes that begin with `01`.
- `NOT FOUND` is shorthand for all `SQLSTATE` codes that begin with `02`.
- `SQLEXCEPTION` is shorthand for all `SQLSTATE` codes not caught by `SQLWARNING` or `NOT FOUND`.

Example:

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> delimiter //
```

```
mysql> CREATE PROCEDURE handlerdemo ()
-> BEGIN
->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
->   SET @x = 1;
->   INSERT INTO test.t VALUES (1);
->   SET @x = 2;
->   INSERT INTO test.t VALUES (1);
->   SET @x = 3;
-> END;
-> //
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CALL handlerdemo();//
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @x//
+-----+
| @x    |
+-----+
| 3     |
+-----+
1 row in set (0.00 sec)
```

The example associates a handler with SQLSTATE 23000, which occurs for a duplicate-key error. Notice that @x is 3, which shows that MySQL executed to the end of the procedure. If the line `DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;` had not been present, MySQL would have taken the default path (EXIT) after the second `INSERT` failed due to the `PRIMARY KEY` constraint, and `SELECT @x` would have returned 2.

If you want to ignore a condition, you can declare a `CONTINUE` handler for it and associate it with an empty block. For example:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

17.2.9. Cursors

Simple cursors are supported inside stored procedures and functions. The syntax

is as in embedded SQL. Cursors are currently asensitive, read-only, and non-scrolling. Asensitive means that the server may or may not make a copy of its result table.

Cursors must be declared before declaring handlers, and variables and conditions must be declared before declaring either cursors or handlers.

Example:

```
CREATE PROCEDURE curdemo()  
BEGIN  
  DECLARE done INT DEFAULT 0;  
  DECLARE a CHAR(16);  
  DECLARE b,c INT;  
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;  
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;  
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;  
  
  OPEN cur1;  
  OPEN cur2;  
  
  REPEAT  
    FETCH cur1 INTO a, b;  
    FETCH cur2 INTO c;  
    IF NOT done THEN  
      IF b < c THEN  
        INSERT INTO test.t3 VALUES (a,b);  
      ELSE  
        INSERT INTO test.t3 VALUES (a,c);  
      END IF;  
    END IF;  
  UNTIL done END REPEAT;  
  
  CLOSE cur1;  
  CLOSE cur2;  
END
```

17.2.9.1. Declaring Cursors

```
DECLARE cursor_name CURSOR FOR select_statement
```

This statement declares a cursor. Multiple cursors may be declared in a routine, but each cursor in a given block must have a unique name.

The SELECT statement cannot have an INTO clause.

17.2.9.2. Cursor OPEN Statement

```
OPEN cursor_name
```

This statement opens a previously declared cursor.

17.2.9.3. Cursor FETCH Statement

```
FETCH cursor_name INTO var_name [, var_name] ...
```

This statement fetches the next row (if a row exists) using the specified open cursor, and advances the cursor pointer.

17.2.9.4. Cursor CLOSE Statement

```
CLOSE cursor_name
```

This statement closes a previously opened cursor.

If not closed explicitly, a cursor is closed at the end of the compound statement in which it was declared.

17.2.10. Flow Control Constructs

The IF, CASE, LOOP, WHILE, REPLACE ITERATE, and LEAVE constructs are fully implemented.

Many of these constructs contain other statements, as indicated by the grammar specifications in the following sections. Such constructs may be nested. For example, an IF statement might contain a WHILE loop, which itself contains a CASE statement.

FOR loops are not currently supported.

17.2.10.1. IF Statement

```
IF search_condition THEN statement_list  
    [ELSEIF search_condition THEN statement_list] ...  
    [ELSE statement_list]  
END IF
```

IF implements a basic conditional construct. If the *search_condition* evaluates to true, the corresponding SQL statement list is executed. If no *search_condition* matches, the statement list in the ELSE clause is executed. Each *statement_list* consists of one or more statements.

Note: There is also an IF() *function*, which differs from the IF *statement* described here. See [Section 12.2, “Control Flow Functions”](#).

17.2.10.2. CASE Statement

```
CASE case_value
  WHEN when_value THEN statement_list
  [WHEN when_value THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

Or:

```
CASE
  WHEN search_condition THEN statement_list
  [WHEN search_condition THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

The CASE statement for stored routines implements a complex conditional construct. If a *search_condition* evaluates to true, the corresponding SQL statement list is executed. If no search condition matches, the statement list in the ELSE clause is executed. Each *statement_list* consists of one or more statements.

Note: The syntax of the CASE *statement* shown here for use inside stored routines differs slightly from that of the SQL CASE *expression* described in [Section 12.2, “Control Flow Functions”](#). The CASE statement cannot have an ELSE NULL clause, and it is terminated with END CASE instead of END.

17.2.10.3. LOOP Statement

```
[begin_label:] LOOP
  statement_list
END LOOP [end_label]
```

LOOP implements a simple loop construct, enabling repeated execution of the

statement list, which consists of one or more statements. The statements within the loop are repeated until the loop is exited; usually this is accomplished with a LEAVE statement.

A LOOP statement can be labeled. *end_label* cannot be given unless *begin_label* also is present. If both are present, they must be the same.

17.2.10.4. LEAVE Statement

LEAVE *label*

This statement is used to exit any labeled flow control construct. It can be used within BEGIN . . . END or loop constructs (LOOP, REPEAT, WHILE).

17.2.10.5. ITERATE Statement

ITERATE *label*

ITERATE can appear only within LOOP, REPEAT, and WHILE statements. ITERATE means “do the loop again.”

Example:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END
```

17.2.10.6. REPEAT Statement

```
[begin_label:] REPEAT
  statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a REPEAT statement is repeated until the *search_condition* is true. Thus, a REPEAT always enters the loop at least once.

statement_list consists of one or more statements.

A REPEAT statement can be labeled. *end_label* cannot be given unless *begin_label* also is present. If both are present, they must be the same.

Example:

```
mysql> delimiter //
```

```
mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
-> SET @x = 0;
-> REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
-> END
-> //
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @x//
+-----+
| @x    |
+-----+
| 1001  |
+-----+
1 row in set (0.00 sec)
```

17.2.10.7. WHILE Statement

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

The statement list within a WHILE statement is repeated as long as the *search_condition* is true. *statement_list* consists of one or more statements.

A WHILE statement can be labeled. *end_label* cannot be given unless *begin_label* also is present. If both are present, they must be the same.

Example:

```
CREATE PROCEDURE dowhile()
BEGIN
    DECLARE v1 INT DEFAULT 5;
```

```
WHILE v1 > 0 DO
  ...
  SET v1 = v1 - 1;
END WHILE;
END
```

17.3. Stored Procedures, Functions, Triggers, and Replication: Frequently Asked Questions

- Do MySQL 5.0 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a master MySQL server to a slave server. There are a few limitations that are described in detail in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

- Are stored procedures and functions created on a master server replicated to a slave?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a master server are replicated to a slave, so the objects will exist on both servers. ALTER and DROP statements for stored procedures and functions are also replicated.

- How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a slave server. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

- Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a slave server has authority to execute any statement read from a master's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active, then MySQL DBAs have two security options open to them:

- Any user wishing to create stored functions must be granted the SUPER

privilege.

- Alternatively, a DBA can set the `log_bin_trust_function_creators` system variable to 1, which enables anyone with the standard `CREATE ROUTINE` privilege to create stored functions.

Note: Before MySQL 5.0.16, these restrictions also apply to stored procedures and the system variable is named `log_bin_trust_routine_creators`.

- What limitations exist for replicating stored procedure and function actions?

Non-deterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced, and therefore, random actions replicated to a slave will not mirror those performed on a master. Note that declaring stored functions to be `DETERMINISTIC` or setting the `log_bin_trust_function_creators` system variable to 0 will not allow random-valued operations to be invoked.

In addition, time-based actions cannot be reproduced on a slave because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, non-transactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a master may be partially updated from DML activity, but no updates are done to the slave because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the `IGNORE` keyword so that updates on the master that cause errors are ignored and updates that do not cause errors are replicated to the slave.

- Do the preceding limitations affect MySQL's ability to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

- What will MySQL do to correct the aforementioned limitations?

A future release of MySQL is expected to feature a choice in how replication should be handled:

- Statement-based replication (current implementation).
 - Row-level replication (that will solve all the limitations described earlier).
- Do triggers work with replication?

Triggers and replication in MySQL 5.0 work the same as in most other database engines: Actions carried out through triggers on a master are not replicated to a slave server. Instead, triggers that exist on tables that reside on a MySQL master server need to be created on the corresponding tables on any MySQL slave servers so that the triggers activate on the slaves as well as the master.

- How are actions carried out through triggers on a master replicated to a slave?

First, the triggers that exist on a master must be re-created on the slave server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table EMP that has an AFTER insert trigger, which exists on a master MySQL server. The same EMP table and AFTER insert trigger exist on the slave server as well. The replication flow would be:

1. An INSERT statement is made to EMP.
2. The AFTER trigger on EMP activates.
3. The INSERT statement is written to the binary log.
4. The replication slave picks up the INSERT statement to EMP and executes it.
5. The AFTER trigger on EMP that exists on the slave activates.

17.4. Binary Logging of Stored Routines and Triggers

The binary log contains information about SQL statements that modify database contents. This information is stored in the form of “events” that describe the modifications. The binary log has two important purposes:

- For replication, the master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See [Section 6.2, “Replication Implementation Overview”](#).
- Certain data recovery operations require use of the binary log. After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See [Section 5.10.2.2, “Using Backups for Recovery”](#).

This section describes the development of binary logging in MySQL 5.0 with respect to stored routines (procedures and functions) and triggers. The discussion first summarizes the changes that have taken place in the logging implementation, and then states the current conditions that the implementation places on the use of stored routines. Finally, implementation details are given that provide information about when and why various changes were made. These details show how several aspects of the current logging behavior were implemented in response to shortcomings identified in earlier versions.

In general, the issues described here result from the fact that binary logging occurs at the SQL statement level. A future MySQL release is expected to implement row-level binary logging, which specifies the changes to make to individual rows as a result of executing SQL statements.

Unless noted otherwise, the remarks here assume that you have enabled binary logging by starting the server with the `--log-bin` option. (See [Section 5.12.3, “The Binary Log”](#).) If the binary log is not enabled, replication is not possible, nor is the binary log available for data recovery.

The development of stored routine logging in MySQL 5.0 can be summarized as follows:

- Before MySQL 5.0.6: In the initial implementation of stored routine logging, statements that create stored routines and CALL statements are not logged. These omissions can cause problems for replication and data recovery.
- MySQL 5.0.6: Statements that create stored routines and CALL statements are logged. Stored function invocations are logged when they occur in statements that update data (because those statements are logged). However, function invocations are not logged when they occur in statements such as SELECT that do not change data, even if a data change occurs within a function itself; this can cause problems. Under some circumstances, functions and procedures can have different effects if executed at different times or on different (master and slave) machines, and thus can be unsafe for data recovery or replication. To handle this, measures are implemented to allow identification of safe routines and to prevent creation of unsafe routines except by users with sufficient privileges.
- MySQL 5.0.12: For stored functions, when a function invocation that changes data occurs within a non-logged statement such as SELECT, the server logs a DO func_name() statement that invokes the function so that the function gets executed during data recovery or replication to slave servers. For stored procedures, the server does not log CALL statements. Instead, it logs individual statements within a procedure that are executed as a result of a CALL. This eliminates problems that may occur when a procedure would follow a different execution path on a slave than on the master.
- MySQL 5.0.16: The procedure logging changes made in 5.0.12 allow the conditions on unsafe routines to be relaxed for stored procedures. Consequently, the user interface for controlling these conditions is revised to apply only to functions. Procedure creators are no longer bound by them.
- MySQL 5.0.17: Logging of stored functions as DO func_name() statements (per the changes made in 5.0.12) are logged as SELECT func_name() statements instead for better control over error checking.

As a consequence of the preceding changes, the following conditions currently apply to stored function creation when binary logging is enabled. These conditions do not apply to stored procedure creation.

- To create or alter a stored function, you must have the `SUPER` privilege, in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege that is normally required.
- When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for data recovery or replication. Two sets of function characteristics apply here:
 - The `DETERMINISTIC` and `NOT DETERMINISTIC` characteristics indicate whether a function always produces the same result for given inputs. The default is `NOT DETERMINISTIC` if neither characteristic is given, so you must specify `DETERMINISTIC` explicitly to declare that a function is deterministic.

Use of the `NOW()` function (or its synonyms) or `RAND()` does not necessarily make a function non-deterministic. For `NOW()`, the binary log includes the timestamp and replicates correctly. `RAND()` also replicates correctly as long as it is invoked only once within a function. (You can consider the function execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

`SYSDATE()` is not affected by the timestamps in the binary log, so it causes stored routines to be non-deterministic if statement-based logging is used. This does not occur if the server is started with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`.

- The `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` characteristics provide information about whether the function reads or writes data. Either `NO SQL` or `READS SQL DATA` indicates that a function does not change data, but you must specify one of these explicitly because the default is `CONTAINS SQL` if no characteristic is given.

By default, for a `CREATE FUNCTION` statement to be accepted, `DETERMINISTIC` or one of `NO SQL` and `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO
```

or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe `log_bin_trust_function_creators` variable)

Assessment of the nature of a function is based on the “honesty” of the creator: MySQL does not check that a function declared `DETERMINISTIC` contains no statements that produce non-deterministic results.

- To relax the preceding conditions on function creation (that you must have the `SUPER` privilege and that a function must be declared deterministic or to not modify data), set the global `log_bin_trust_function_creators` system variable to 1. By default, this variable has a value of 0, but you can change it like this:

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

You can also set this variable by using the `--log-bin-trust-function-creators` option when starting the server.

If binary logging is not enabled, `log_bin_trust_function_creators` does not apply and `SUPER` is not required for routine creation.

Triggers are similar to stored functions, so the preceding remarks regarding functions also apply to triggers with the following exception: `CREATE TRIGGER` does not have an optional `DETERMINISTIC` characteristic, so triggers are assumed to be always deterministic. However, this assumption might in some cases be invalid. For example, the `UUID()` function is non-deterministic (and does not replicate). You should be careful about using such functions in triggers.

Triggers can update tables (as of MySQL 5.0.10), so error messages similar to those for stored functions occur with `CREATE TRIGGER` if you do not have the `SUPER` privilege and `log_bin_trust_function_creators` is 0.

The rest of this section provides details on the development of stored routine logging. Some of these details give additional background on the rationale for the current logging-related conditions on stored routine use.

Routine logging before MySQL 5.0.6: Statements that create and use stored routines are not written to the binary log, but statements invoked within stored routines are logged. Suppose that you issue the following statements:

```
CREATE PROCEDURE mysp INSERT INTO t VALUES(1);  
CALL mysp();
```

For this example, only the `INSERT` statement appears in the binary log. The `CREATE PROCEDURE` and `CALL` statements do not appear. The absence of routine-related statements in the binary log means that stored routines are not replicated correctly. It also means that for a data recovery operation, re-executing events in the binary log does not recover stored routines.

Routine logging changes in MySQL 5.0.6: To address the absence of logging for stored routine creation and `CALL` statements (and the consequent replication and data recovery concerns), the characteristics of binary logging for stored routines were changed as described here. (Some of the items in the following list point out issues that are dealt with in later versions.)

- The server writes `CREATE PROCEDURE`, `CREATE FUNCTION`, `ALTER PROCEDURE`, `ALTER FUNCTION`, `DROP PROCEDURE`, and `DROP FUNCTION` statements to the binary log. Also, the server logs `CALL` statements, not the statements executed within procedures. Suppose that you issue the following statements:

```
CREATE PROCEDURE mysp INSERT INTO t VALUES(1);  
CALL mysp();
```

For this example, the `CREATE PROCEDURE` and `CALL` statements appear in the binary log, but the `INSERT` statement does not appear. This corrects the problem that occurred before MySQL 5.0.6 such that only the `INSERT` was logged.

- Logging `CALL` statements has a security implication for replication, which arises from two factors:
 - It is possible for a procedure to follow different execution paths on master and slave servers.
 - Statements executed on a slave are processed by the slave SQL thread which has full privileges.

The implication is that although a user must have the `CREATE ROUTINE` privilege to create a routine, the user can write a routine containing a dangerous statement that will execute only on the slave where the statement

is processed by the SQL thread that has full privileges. For example, if the master and slave servers have server ID values of 1 and 2, respectively, a user on the master server could create and invoke an unsafe procedure `unsafe_sp()` as follows:

```
mysql> delimiter //
mysql> CREATE PROCEDURE unsafe_sp ()
-> BEGIN
->   IF @@server_id=2 THEN DROP DATABASE accounting; END IF;
-> END;
-> //
mysql> delimiter ;
mysql> CALL unsafe_sp();
```

The `CREATE PROCEDURE` and `CALL` statements are written to the binary log, so the slave will execute them. Because the slave SQL thread has full privileges, it will execute the `DROP DATABASE` statement that drops the accounting database. Thus, the `CALL` statement has different effects on the master and slave and is not replication-safe.

The preceding example uses a stored procedure, but similar problems can occur for stored functions that are invoked within statements that are written to the binary log: Function invocation has different effects on the master and slave.

To guard against this danger for servers that have binary logging enabled, MySQL 5.0.6 introduces the requirement that stored procedure and function creators must have the `SUPER` privilege, in addition to the usual `CREATE ROUTINE` privilege that is required. Similarly, to use `ALTER PROCEDURE` or `ALTER FUNCTION`, you must have the `SUPER` privilege in addition to the `ALTER ROUTINE` privilege. Without the `SUPER` privilege, an error will occur:

```
ERROR 1419 (HY000): You do not have the SUPER privilege and
binary logging is enabled (you *might* want to use the less safe
log_bin_trust_routine_creators variable)
```

If you do not want to require routine creators to have the `SUPER` privilege (for example, if all users with the `CREATE ROUTINE` privilege on your system are experienced application developers), set the global `log_bin_trust_routine_creators` system variable to 1. You can also set this variable by using the `--log-bin-trust-routine-creators` option when starting the server. If binary logging is not enabled,

`log_bin_trust_routine_creators` does not apply and `SUPER` is not required for routine creation.

- If a routine that performs updates is non-deterministic, it is not repeatable. This can have two undesirable effects:
 - It will make a slave different from the master.
 - Restored data will be different from the original data.

To deal with these problems, MySQL enforces the following requirement: On a master server, creation and alteration of a routine is refused unless you declare the routine to be deterministic or to not modify data. Two sets of routine characteristics apply here:

- The `DETERMINISTIC` and `NOT DETERMINISTIC` characteristics indicate whether a routine always produces the same result for given inputs. The default is `NOT DETERMINISTIC` if neither characteristic is given, so you must specify `DETERMINISTIC` explicitly to declare that a routine is deterministic.
- The `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` characteristics provide information about whether the routine reads or writes data. Either `NO SQL` or `READS SQL DATA` indicates that a routine does not change data, but you must specify one of these explicitly because the default is `CONTAINS SQL` if no characteristic is given.

By default, for a `CREATE PROCEDURE` or `CREATE FUNCTION` statement to be accepted, `DETERMINISTIC` or one of `NO SQL` and `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This routine has none of DETERMINISTIC, NO SQL or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_routine_creators variable)
```

If you set `log_bin_trust_routine_creators` to 1, the requirement that routines be deterministic or not modify data is dropped.

- A `CALL` statement is written to the binary log if the routine returns no error,

but not otherwise. When a routine that modifies data fails, you get this warning:

```
ERROR 1417 (HY000): A routine failed and has neither NO SQL nor
READS SQL DATA in its declaration and binary logging is enabled;
non-transactional tables were updated, the binary log will miss
changes
```

This logging behavior has the potential to cause problems. If a routine partly modifies a non-transactional table (such as a MyISAM table) and returns an error, the binary log will not reflect these changes. To protect against this, you should use transactional tables in the routine and modify the tables within transactions.

If you use the IGNORE keyword with INSERT, DELETE, or UPDATE to ignore errors within a routine, a partial update might occur but no error will result. Such statements are logged and they replicate normally.

- Although statements normally are not written to the binary log if they are rolled back, CALL statements are logged even when they occur within a rolled-back transaction. This can result in a CALL being rolled back on the master but executed on slaves.
- If a stored function is invoked within a statement such as SELECT that does not modify data, execution of the function is not written to the binary log, even if the function itself modifies data. This logging behavior has the potential to cause problems. Suppose that a function myfunc() is defined as follows:

```
CREATE FUNCTION myfunc () RETURNS INT DETERMINISTIC
BEGIN
  INSERT INTO t (i) VALUES(1);
  RETURN 0;
END;
```

Given that definition, the following statement is not written to the binary log because it is a SELECT. Nevertheless, it modifies the table t because myfunc() modifies t:

```
SELECT myfunc();
```

A workaround for this problem is to invoke functions that do updates only

within statements that do updates (and which therefore are written to the binary log). Note that although the `DO` statement sometimes is executed for the side effect of evaluating an expression, `DO` is not a workaround here because it is not written to the binary log.

- On slave servers, `--replicate-*table` rules do not apply to `CALL` statements or to statements within stored routines. These statements are always replicated. If such statements contain references to tables that do not exist on the slave, they could have undesirable effects when executed on the slave.

Routine logging changes in MySQL 5.0.12: The changes in 5.0.12 address several problems that were present in earlier versions:

- Stored function invocations in non-logged statements such as `SELECT` were not being logged, even when a function itself changed data.
- Stored procedure logging at the `CALL` level could cause different effects on a master and slave if a procedure took different execution paths on the two machines.
- `CALL` statements were logged even when they occurred within a rolled-back transaction.

To deal with these issues, MySQL 5.0.12 implements the following changes to function and procedure logging:

- A stored function invocation is logged as a `DO` statement if the function changes data and occurs within a statement that would not otherwise be logged. This corrects the problem of non-replication of data changes that result from use of stored functions in non-logged statements. For example, `SELECT` statements are not written to the binary log, but a `SELECT` might invoke a stored function that makes changes. To handle this, a `DO func_name()` statement is written to the binary log when the given function makes a change. Suppose that the following statements are executed on the master:

```
CREATE FUNCTION f1(a INT) RETURNS INT
BEGIN
  IF (a < 3) THEN
    INSERT INTO t2 VALUES (a);
```

```
    END IF;
END;

CREATE TABLE t1 (a INT);
INSERT INTO t1 VALUES (1),(2),(3);

SELECT f1(a) FROM t1;
```

When the `SELECT` statement executes, the function `f1()` is invoked three times. Two of those invocations insert a row, and MySQL logs a `DO` statement for each of them. That is, MySQL writes the following statements to the binary log:

```
DO f1(1);
DO f1(2);
```

The server also logs a `DO` statement for a stored function invocation when the function invokes a stored procedure that causes an error. In this case, the server writes the `DO` statement to the log along with the expected error code. On the slave, if the same error occurs, that is the expected result and replication continues. Otherwise, replication stops.

Note: See later in this section for changes made in MySQL 5.0.19: These logged `DO func_name()` statements are logged as `SELECT func_name()` statements instead.

- Stored procedure calls are logged at the statement level rather than at the `CALL` level. That is, the server does not log the `CALL` statement, it logs those statements within the procedure that actually execute. As a result, the same changes that occur on the master will be observed on slave servers. This eliminates the problems that could result from a procedure having different execution paths on different machines. For example, the `DROP DATABASE` problem shown earlier for the `unsafe_sp()` procedure does not occur and the routine is no longer replication-unsafe because it has the same effect on master and slave servers.

In general, statements executed within a stored procedure are written to the binary log using the same rules that would apply were the statements to be executed in standalone fashion. Some special care is taken when logging procedure statements because statement execution within procedures is not quite the same as in non-procedure context:

- A statement to be logged might contain references to local procedure variables. These variables do not exist outside of stored procedure context, so a statement that refers to such a variable cannot be logged literally. Instead, each reference to a local variable is replaced by this construct for logging purposes:

```
NAME_CONST(var_name, var_value)
```

var_name is the local variable name, and *var_value* is a constant indicating the value that the variable has at the time the statement is logged. `NAME_CONST()` has a value of *var_value*, and a “name” of *var_name*. Thus, if you invoke this function directly, you get a result like this:

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|      14 |
+-----+
```

`NAME_CONST()` allows a logged standalone statement to be executed on a slave with the same effect as the original statement that was executed on the master within a stored procedure.

- A statement to be logged might contain references to user-defined variables. To handle this, MySQL writes a `SET` statement to the binary log to make sure that the variable exists on the slave with the same value as on the master. For example, if a statement refers to a variable `@my_var`, that statement will be preceded in the binary log by the following statement, where *value* is the value of `@my_var` on the master:

```
SET @my_var = value;
```

- Procedure calls can occur within a committed or rolled-back transaction. Previously, `CALL` statements were logged even if they occurred within a rolled-back transaction. As of MySQL 5.0.12, transactional context is accounted for so that the transactional aspects of procedure execution are replicated correctly. That is, the server logs those statements within the procedure that actually execute and modify

data, and also logs BEGIN, COMMIT, and ROLLBACK statements as necessary. For example, if a procedure updates only transactional tables and is executed within a transaction that is rolled back, those updates are not logged. If the procedure occurs within a committed transaction, BEGIN and COMMIT statements are logged with the updates. For a procedure that executes within a rolled-back transaction, its statements are logged using the same rules that would apply if the statements were executed in standalone fashion:

- Updates to transactional tables are not logged.
 - Updates to non-transactional tables are logged because rollback does not cancel them.
 - Updates to a mix of transactional and non-transactional tables are logged surrounded by BEGIN and ROLLBACK so that slaves will make the same changes and rollbacks as on the master.
- A stored procedure call is *not* written to the binary log at the statement level if the procedure is invoked from within a stored function. In that case, the only thing logged is the statement that invokes the function (if it occurs within a statement that is logged) or a DO statement (if it occurs within a statement that is not logged). For this reason, care still should be exercised in the use of stored functions that invoke a procedure, even if the procedure is otherwise safe in itself.
 - Because procedure logging occurs at the statement level rather than at the CALL level, interpretation of the `--replicate-* -table` options is revised to apply only to stored functions. They no longer apply to stored procedures, except those procedures that are invoked from within functions.

Routine logging changes in MySQL 5.0.16: In 5.0.12, a change was introduced to log stored procedure calls at the statement level rather than at the CALL level. This change eliminates the requirement that procedures be identified as safe. The requirement now exists only for stored functions, because they still appear in the binary log as function invocations rather than as the statements executed within the function. To reflect the lifting of the restriction on stored procedures, the `log_bin_trust_routine_creators` system variable is renamed to `log_bin_trust_function_creators` and the `--log-bin-trust-routine-`

creators server option is renamed to `--log-bin-trust-function-creators`. (For backward compatibility, the old names are recognized but result in a warning.) Error messages that now apply only to functions and not to routines in general are re-worded.

Routine logging changes in MySQL 5.0.19: In 5.0.12, a change was introduced to log a stored function invocation as `DO func_name()` if the invocation changes data and occurs within a non-logged statement, or if the function invokes a stored procedure that produces an error. In 5.0.19, these invocations are logged as `SELECT func_name()` instead. The change to `SELECT` was made because use of `DO` was found to yield insufficient control over error code checking.

Chapter 18. Triggers

Table of Contents

[18.1. CREATE TRIGGER Syntax](#)

[18.2. DROP TRIGGER Syntax](#)

[18.3. Using Triggers](#)

Support for triggers is included beginning with MySQL 5.0.2. A trigger is a named database object that is associated with a table and that is activated when a particular event occurs for the table. For example, the following statements create a table and an INSERT trigger. The trigger sums the values inserted into one of the table's columns:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));  
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account  
-> FOR EACH ROW SET @sum = @sum + NEW.amount;  
Query OK, 0 rows affected (0.06 sec)
```

This chapter describes the syntax for creating and dropping triggers, and shows some examples of how to use them. Discussion of restrictions on use of triggers is given in [Section I.1, “Restrictions on Stored Routines and Triggers”](#). Remarks regarding binary logging as it applies to triggers are given in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

18.1. CREATE TRIGGER Syntax

```
CREATE
  [DEFINER = { user | CURRENT_USER }]
  TRIGGER trigger_name trigger_time trigger_event
  ON tbl_name FOR EACH ROW trigger_stmt
```

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. CREATE TRIGGER was added in MySQL 5.0.2. Currently, its use requires the SUPER privilege.

The trigger becomes associated with the table named *tbl_name*, which must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view.

When the trigger is activated, the DEFINER clause determines the privileges that apply, as described later in this section.

trigger_time is the trigger action time. It can be BEFORE or AFTER to indicate that the trigger activates before or after the statement that activated it.

trigger_event indicates the kind of statement that activates the trigger. The *trigger_event* can be one of the following:

- INSERT: The trigger is activated whenever a new row is inserted into the table; for example, through INSERT, LOAD DATA, and REPLACE statements.
- UPDATE: The trigger is activated whenever a row is modified; for example, through UPDATE statements.
- DELETE: The trigger is activated whenever a row is deleted from the table; for example, through DELETE and REPLACE statements. However, DROP TABLE and TRUNCATE statements on the table do *not* activate this trigger, because they do not use DELETE. See [Section 13.2.9, “TRUNCATE Syntax”](#).

It is important to understand that the *trigger_event* does not represent a literal type of SQL statement that activates the trigger so much as it represents a type of table operation. For example, an INSERT trigger is activated by not only INSERT

statements but also LOAD DATA statements because both statements insert rows into a table.

A potentially confusing example of this is the INSERT INTO ... ON DUPLICATE KEY UPDATE ... syntax: a BEFORE INSERT trigger will activate for every row, followed by either an AFTER INSERT trigger or both the BEFORE UPDATE and AFTER UPDATE triggers, depending on whether there was a duplicate key for the row.

There cannot be two triggers for a given table that have the same trigger action time and event. For example, you cannot have two BEFORE UPDATE triggers for a table. But you can have a BEFORE UPDATE and a BEFORE INSERT trigger, or a BEFORE UPDATE and an AFTER UPDATE trigger.

trigger_stmt is the statement to execute when the trigger activates. If you want to execute multiple statements, use the BEGIN ... END compound statement construct. This also enables you to use the same statements that are allowable within stored routines. See [Section 17.2.5, “BEGIN ... END Compound Statement Syntax”](#). Some statements are not allowed in triggers; see [Section I.1, “Restrictions on Stored Routines and Triggers”](#).

Note: Currently, triggers are not activated by cascaded foreign key actions. This limitation will be lifted as soon as possible.

Note: Before MySQL 5.0.10, triggers cannot contain direct references to tables by name. Beginning with MySQL 5.0.10, you can write triggers such as the one named testref shown in this example:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);
```

```
DELIMITER |
```

```
CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW BEGIN
  INSERT INTO test2 SET a2 = NEW.a1;
  DELETE FROM test3 WHERE a3 = NEW.a1;
  UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
```

```

    END;
|
DELIMITER ;

INSERT INTO test3 (a3) VALUES
    (NULL), (NULL), (NULL), (NULL), (NULL),
    (NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
    (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);

```

Suppose that you insert the following values into table test1 as shown here:

```

mysql> INSERT INTO test1 VALUES
    -> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0

```

As a result, the data in the four tables will be as follows:

```

mysql> SELECT * FROM test1;
+-----+
| a1    |
+-----+
| 1     |
| 3     |
| 1     |
| 7     |
| 1     |
| 8     |
| 4     |
| 4     |
+-----+
8 rows in set (0.00 sec)

```

```

mysql> SELECT * FROM test2;
+-----+
| a2    |
+-----+
| 1     |
| 3     |
| 1     |
| 7     |
| 1     |
| 8     |
| 4     |
| 4     |
+-----+

```

8 rows in set (0.00 sec)

```
mysql> SELECT * FROM test3;
```

```
+-----+
| a3 |
+-----+
|  2 |
|  5 |
|  6 |
|  9 |
| 10 |
+-----+
```

5 rows in set (0.00 sec)

```
mysql> SELECT * FROM test4;
```

```
+-----+-----+
| a4 | b4 |
+-----+-----+
|  1 |  3 |
|  2 |  0 |
|  3 |  1 |
|  4 |  2 |
|  5 |  0 |
|  6 |  0 |
|  7 |  1 |
|  8 |  1 |
|  9 |  0 |
| 10 |  0 |
+-----+-----+
```

10 rows in set (0.00 sec)

You can refer to columns in the subject table (the table associated with the trigger) by using the aliases `OLD` and `NEW`. `OLD.col_name` refers to a column of an existing row before it is updated or deleted. `NEW.col_name` refers to the column of a new row to be inserted or an existing row after it is updated.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at trigger activation time. It was added in MySQL 5.0.17. If a *user* value is given, it should be a MySQL account in '*user_name*'@'*host_name*' format (the same format used in the `GRANT` statement). The *user_name* and *host_name* values both are required. `CURRENT_USER` also can be given as `CURRENT_USER()`. The default `DEFINER` value is the user who executes the `CREATE TRIGGER` statement. (This is the same as `DEFINER = CURRENT_USER`.)

If you specify the `DEFINER` clause, you cannot set the value to any account but your own unless you have the `SUPER` privilege. These rules determine the legal

DEFINER user values:

- If you do not have the SUPER privilege, the only legal *user* value is your own account, either specified literally or by using CURRENT_USER. You cannot set the definer to some other account.
- If you have the SUPER privilege, you can specify any syntactically legal account name. If the account does not actually exist, a warning is generated.

Although it is possible to create triggers with a non-existent DEFINER value, it is not a good idea for such triggers to be activated until the definer actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.

Note: Because MySQL currently requires the SUPER privilege for the use of CREATE TRIGGER, only the second of the preceding rules applies. (MySQL 5.1.6 implements the TRIGGER privilege and requires that privilege for trigger creation, so at that point both rules come into play and SUPER is required only for specifying a DEFINER value other than your own account.)

From MySQL 5.0.17 on, MySQL checks trigger privileges like this:

- At CREATE TRIGGER time, the user that issues the statement must have the SUPER privilege.
- At trigger activation time, privileges are checked against the DEFINER user. This user must have these privileges:
 - The SUPER privilege.
 - The SELECT privilege for the subject table if references to table columns occur via OLD.col_name or NEW.col_name in the trigger definition.
 - The UPDATE privilege for the subject table if table columns are targets of SET NEW.col_name = value assignments in the trigger definition.
 - Whatever other privileges normally are required for the statements executed by the trigger.

Before MySQL 5.0.17, MySQL checks trigger privileges like this:

- At `CREATE TRIGGER` time, the user that issues the statement must have the `SUPER` privilege.
- At trigger activation time, privileges are checked against the user whose actions cause the trigger to be activated. This user must have whatever privileges normally are required for the statements executed by the trigger.

Note that the introduction of the `DEFINER` clause changes the meaning of `CURRENT_USER()` within trigger definitions: The `CURRENT_USER()` function evaluates to the trigger `DEFINER` value as of MySQL 5.0.17 and to the user whose actions caused the trigger to be activated before 5.0.17.

18.2. DROP TRIGGER Syntax

```
DROP TRIGGER [schema_name.] trigger_name
```

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. `DROP TRIGGER` was added in MySQL 5.0.2. Its use requires the `SUPER` privilege.

Note: Prior to MySQL 5.0.10, the table name was required instead of the schema name (*table_name.trigger_name*). When upgrading from a previous version of MySQL 5.0 to MySQL 5.0.10 or newer, you must drop all triggers *before upgrading* and re-create them afterwards, or else `DROP TRIGGER` does not work after the upgrade. See [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#), for a suggested upgrade procedure.

In addition, triggers created in MySQL 5.0.16 or later cannot be dropped following a downgrade to MySQL 5.0.15 or earlier. If you wish to perform such a downgrade, you must also in this case drop all triggers *prior to* the downgrade, and then re-create them afterwards.

(For more information about these two issues, see Bug #15921 and Bug #18588.)

18.3. Using Triggers

Support for triggers is included beginning with MySQL 5.0.2. This section discusses how to use triggers and some limitations regarding their use. Additional information about trigger limitations is given in [Section I.1, “Restrictions on Stored Routines and Triggers”](#).

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is associated with a table and is defined to activate when an INSERT, DELETE, or UPDATE statement for the table executes. A trigger can be set to activate either before or after the triggering statement. For example, you can have a trigger activate before each row that is deleted from a table or after each row that is updated.

To create a trigger or drop a trigger, use the CREATE TRIGGER or DROP TRIGGER statement. The syntax for these statements is described in [Section 18.1, “CREATE TRIGGER Syntax”](#), and [Section 18.2, “DROP TRIGGER Syntax”](#).

Here is a simple example that associates a trigger with a table for INSERT statements. It acts as an accumulator to sum the values inserted into one of the columns of the table.

The following statements create a table and a trigger for it:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

The CREATE TRIGGER statement creates a trigger named ins_sum that is associated with the account table. It also includes clauses that specify the trigger activation time, the triggering event, and what to do with the trigger activates:

- The keyword BEFORE indicates the trigger action time. In this case, the trigger should activate before each row inserted into the table. The other allowable keyword here is AFTER.

- The keyword `INSERT` indicates the event that activates the trigger. In the example, `INSERT` statements cause trigger activation. You can also create triggers for `DELETE` and `UPDATE` statements.
- The statement following `FOR EACH ROW` defines the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering statement. In the example, the triggered statement is a simple `SET` that accumulates the values inserted into the amount column. The statement refers to the column as `NEW.amount` which means “the value of the amount column to be inserted into the new row.”

To use the trigger, set the accumulator variable to zero, execute an `INSERT` statement, and then see what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98), (141,1937.50), (97, -100.
mysql> SELECT @sum AS 'Total amount inserted';
+-----+
| Total amount inserted |
+-----+
| 1852.48                |
+-----+
```

In this case, the value of `@sum` after the `INSERT` statement has executed is `14.98 + 1937.50 - 100`, or `1852.48`.

To destroy the trigger, use a `DROP TRIGGER` statement. You must specify the schema name if the trigger is not in the default schema:

```
mysql> DROP TRIGGER test.ins_sum;
```

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

In addition to the requirement that trigger names be unique for a schema, there are other limitations on the types of triggers you can create. In particular, you cannot have two triggers for a table that have the same activation time and activation event. For example, you cannot define two `BEFORE INSERT` triggers or two `AFTER UPDATE` triggers for a table. This should rarely be a significant limitation, because it is possible to define a trigger that executes multiple statements by using the `BEGIN . . . END` compound statement construct after `FOR`

EACH ROW. (An example appears later in this section.)

The OLD and NEW keywords enable you to access columns in the rows affected by a trigger. (OLD and NEW are not case sensitive.) In an INSERT trigger, only NEW.col_name can be used; there is no old row. In a DELETE trigger, only OLD.col_name can be used; there is no new row. In an UPDATE trigger, you can use OLD.col_name to refer to the columns of a row before it is updated and NEW.col_name to refer to the columns of the row after it is updated.

A column named with OLD is read-only. You can refer to it (if you have the SELECT privilege), but not modify it. A column named with NEW can be referred to if you have the SELECT privilege for it. In a BEFORE trigger, you can also change its value with SET NEW.col_name = *value* if you have the UPDATE privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or that are used to update a row.

In a BEFORE trigger, the NEW value for an AUTO_INCREMENT column is 0, not the automatically generated sequence number that will be generated when the new record actually is inserted.

OLD and NEW are MySQL extensions to triggers.

By using the BEGIN . . . END construct, you can define a trigger that executes multiple statements. Within the BEGIN block, you also can use other syntax that is allowed within stored routines such as conditionals and loops. However, just as for stored routines, if you use the **mysql** program to define a trigger that executes multiple statements, it is necessary to redefine the **mysql** statement delimiter so that you can use the ; statement delimiter within the trigger definition. The following example illustrates these points. It defines an UPDATE trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a BEFORE trigger because the value needs to be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
-> FOR EACH ROW
-> BEGIN
->     IF NEW.amount < 0 THEN
->         SET NEW.amount = 0;
->     ELSEIF NEW.amount > 100 THEN
->         SET NEW.amount = 100;
```

```
->     END IF;  
-> END;//  
mysql> delimiter ;
```

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple CALL statement. This is also advantageous if you want to invoke the same routine from within several triggers.

There are some limitations on what can appear in statements that a trigger executes when activated:

- The trigger cannot use the CALL statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are allowed to return data to the trigger through OUT or INOUT parameters.)
- The trigger cannot use statements that explicitly or implicitly begin or end a transaction such as START TRANSACTION, COMMIT, or ROLLBACK.
- Prior to MySQL 5.0.10, triggers cannot contain direct references to tables by name.

MySQL handles errors during trigger execution as follows:

- If a BEFORE trigger fails, the operation on the corresponding row is not performed.
- An AFTER trigger is executed only if the BEFORE trigger (if any) and the row operation both execute successfully.
- An error during either a BEFORE or AFTER trigger results in failure of the entire statement that caused trigger invocation.
- For transactional tables, failure of a trigger (and thus the whole statement) should cause rollback of all changes performed by the statement. For non-transactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

Chapter 19. Views

Table of Contents

[19.1. ALTER VIEW Syntax](#)

[19.2. CREATE VIEW Syntax](#)

[19.3. DROP VIEW Syntax](#)

Views (including updatable views) are implemented in MySQL Server 5.0. Views are available in binary releases from 5.0.1 and up.

This chapter discusses the following topics:

- Creating or altering views with `CREATE VIEW` or `ALTER VIEW`
- Destroying views with `DROP VIEW`

Discussion of restrictions on use of views is given in [Section I.4, “Restrictions on Views”](#).

To use views if you have upgraded to MySQL 5.0.1 from an older release, you should upgrade your grant tables so that they contain the view-related privileges. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

Metadata about views can be obtained from the `INFORMATION_SCHEMA.VIEWS` table and by using the `SHOW CREATE VIEW` statement. See [Section 20.15, “The INFORMATION_SCHEMA VIEWS Table”](#), and [Section 13.5.4.7, “SHOW CREATE VIEW Syntax”](#).

19.1. ALTER VIEW Syntax

ALTER

```
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  
[DEFINER = { user | CURRENT_USER }]  
[SQL SECURITY { DEFINER | INVOKER }]  
VIEW view_name [(column_list)]  
AS select_statement  
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement changes the definition of an existing view. The syntax is similar to that for CREATE VIEW. See [Section 19.2, “CREATE VIEW Syntax”](#). This statement requires the CREATE VIEW and DROP privileges for the view, and some privilege for each column referred to in the SELECT statement.

This statement was added in MySQL 5.0.1. The DEFINER and SQL SECURITY clauses may be used as of MySQL 5.0.16 to specify the security context to be used when checking access privileges at view invocation time. For details, see [Section 19.2, “CREATE VIEW Syntax”](#).

19.2. CREATE VIEW Syntax

```
CREATE
  [OR REPLACE]
  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
  [DEFINER = { user | CURRENT_USER }]
  [SQL SECURITY { DEFINER | INVOKER }]
  VIEW view_name [(column_list)]
  AS select_statement
  [WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement creates a new view, or replaces an existing one if the `OR REPLACE` clause is given. The *select_statement* is a `SELECT` statement that provides the definition of the view. The statement can select from base tables or other views.

This statement requires the `CREATE VIEW` privilege for the view, and some privilege for each column selected by the `SELECT` statement. For columns used elsewhere in the `SELECT` statement you must have the `SELECT` privilege. If the `OR REPLACE` clause is present, you must also have the `DROP` privilege for the view.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, specify the name as *db_name.view_name* when you create it.

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Base tables and views share the same namespace within a database, so a database cannot contain a base table and a view that have the same name.

Views must have unique column names with no duplicates, just like base tables. By default, the names of the columns retrieved by the `SELECT` statement are used for the view column names. To define explicit names for the view columns, the optional *column_list* clause can be given as a list of comma-separated identifiers. The number of names in *column_list* must be the same as the number of columns retrieved by the `SELECT` statement.

Columns retrieved by the `SELECT` statement can be simple references to table columns. They can also be expressions that use functions, constant values, operators, and so forth.

Unqualified table or view names in the `SELECT` statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the proper database name.

A view can be created from many kinds of `SELECT` statements. It can refer to base tables or other views. It can use joins, `UNION`, and subqueries. The `SELECT` need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t
mysql> SELECT * FROM v;
+-----+-----+-----+
| qty  | price | value |
+-----+-----+-----+
|    3 |    50 |   150 |
+-----+-----+-----+
```

A view definition is subject to the following restrictions:

- The `SELECT` statement cannot contain a subquery in the `FROM` clause.
- The `SELECT` statement cannot refer to system or user variables.
- The `SELECT` statement cannot refer to prepared statement parameters.
- Within a stored routine, the definition cannot refer to routine parameters or local variables.
- Any table or view referred to in the definition must exist. However, after a view has been created, it is possible to drop a table or view that the definition refers to. In this case, use of the view results in an error. To check a view definition for problems of this kind, use the `CHECK TABLE` statement.
- The definition cannot refer to a `TEMPORARY` table, and you cannot create a `TEMPORARY` view.
- The tables named in the view definition must already exist.
- You cannot associate a trigger with a view.

ORDER BY is allowed in a view definition, but it is ignored if you select from a view using a statement that has its own ORDER BY.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a LIMIT clause, and you select from the view using a statement that has its own LIMIT clause, it is undefined which limit applies. This same principle applies to options such as ALL, DISTINCT, or SQL_SMALL_RESULT that follow the SELECT keyword, and to clauses such as INTO, FOR UPDATE, LOCK IN SHARE MODE, and PROCEDURE.

If you create a view and then change the query processing environment by changing system variables, that may affect the results that you get from the view:

```
mysql> CREATE VIEW v AS SELECT CHARSET(CHAR(65)), COLLATION(CHAR(65))
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SET NAMES 'latin1';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT * FROM v;
+-----+-----+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-----+-----+
| latin1           | latin1_swedish_ci   |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT * FROM v;
+-----+-----+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-----+-----+
| utf8             | utf8_general_ci     |
+-----+-----+
1 row in set (0.00 sec)
```

The DEFINER and SQL SECURITY clauses specify the security context to be used when checking access privileges at view invocation time. They were added in MySQL 5.0.13, but have no effect until MySQL 5.0.16.

CURRENT_USER also can be given as CURRENT_USER().

Within a stored routine that is defined with the `SQL SECURITY DEFINER` characteristic, `CURRENT_USER` returns the routine creator. This also affects a view defined within such a routine, if the view definition contains a `DEFINER` value of `CURRENT_USER`.

The default `DEFINER` value is the user who executes the `CREATE VIEW` statement. (This is the same as `DEFINER = CURRENT_USER`.) If a *user* value is given, it should be a MySQL account in `'user_name'@'host_name'` format (the same format used in the `GRANT` statement). The *user_name* and *host_name* values both are required.

If you specify the `DEFINER` clause, you cannot set the value to any user but your own unless you have the `SUPER` privilege. These rules determine the legal `DEFINER` user values:

- If you do not have the `SUPER` privilege, the only legal *user* value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.
- If you have the `SUPER` privilege, you can specify any syntactically legal account name. If the account does not actually exist, a warning is generated.

The `SQL SECURITY` characteristic determines which MySQL account to use when checking access privileges for the view when the view is executed. The legal characteristic values are `DEFINER` and `INVOKER`. These indicate that the view must be executable by the user who defined it or invoked it, respectively. The default `SQL SECURITY` value is `DEFINER`.

As of MySQL 5.0.16 (when the `DEFINER` and `SQL SECURITY` clauses were implemented), view privileges are checked like this:

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required when the function runs can be checked only as it executes: For different invocations of the function, different execution paths within the function might be taken.
- At view execution time, privileges for objects accessed by the view are checked against the privileges held by the view creator or invoker,

depending on whether the SQL SECURITY characteristic is DEFINER or INVOKER, respectively.

- If view execution causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function is defined with a SQL SECURITY characteristic of DEFINER or INVOKER. If the security characteristic is DEFINER, the function runs with the privileges of its creator. If the characteristic is INVOKER, the function runs with the privileges determined by the view's SQL SECURITY characteristic.

Prior to MySQL 5.0.16 (before the DEFINER and SQL SECURITY clauses were implemented), privileges required for objects used in a view are checked at view creation time.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function `f()`:

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

Suppose that `f()` contains a statement such as this:

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

The privileges required for executing statements within `f()` need to be checked when `f()` executes. This might mean that privileges are needed for `p1()` or `p2()`, depending on the execution path within `f()`. Those privileges need to be checked at runtime, and the user who must possess the privileges is determined by the SQL SECURITY values of the function `f()` and the view `v`.

The DEFINER and SQL SECURITY clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for SQL SECURITY INVOKER.

If you invoke a view that was created before MySQL 5.0.13, it is treated as though it was created with a SQL SECURITY DEFINER clause and with a DEFINER value that is the same as your account. However, because the actual definer is

unknown, MySQL issues a warning. To make the warning go away, it is sufficient to re-create the view so that the view definition includes a `DEFINER` clause.

The optional `ALGORITHM` clause is a MySQL extension to standard SQL. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`. The default algorithm is `UNDEFINED` if no `ALGORITHM` clause is present. The algorithm affects how MySQL processes the view.

For `MERGE`, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.

For `TEMPTABLE`, the results from the view are retrieved into a temporary table, which then is used to execute the statement.

For `UNDEFINED`, MySQL chooses which algorithm to use. It prefers `MERGE` over `TEMPTABLE` if possible, because `MERGE` is usually more efficient and because a view cannot be updatable if a temporary table is used.

A reason to choose `TEMPTABLE` explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the `MERGE` algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be `UNDEFINED` for three reasons:

- No `ALGORITHM` clause is present in the `CREATE VIEW` statement.
- The `CREATE VIEW` statement has an explicit `ALGORITHM = UNDEFINED` clause.
- `ALGORITHM = MERGE` is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to `UNDEFINED`.

As mentioned earlier, `MERGE` is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the `MERGE` algorithm works. The examples

assume that there is a view `v_merge` that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- `v_merge` becomes `t`
- `*` becomes `vc1, vc2`, which corresponds to `c1, c2`
- The view `WHERE` clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 2: Suppose that we issue this statement:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that `vc1 < 100` becomes `c1 < 100` and the view `WHERE` clause is added to the statement `WHERE` clause using an `AND` connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Effectively, the statement to be executed has a `WHERE` clause of this form:

```
WHERE (select WHERE) AND (view WHERE)
```

The `MERGE` algorithm requires a one-to-one relationship between the rows in the view and the rows in the underlying table. If this relationship does not hold, a temporary table must be used instead. Lack of a one-to-one relationship occurs if the view contains any of a number of constructs:

- Aggregate functions (SUM(), MIN(), MAX(), COUNT(), and so forth)
- DISTINCT
- GROUP BY
- HAVING
- UNION or UNION ALL
- Refers only to literal values (in this case, there is no underlying table)

Some views are updatable. That is, you can use them in statements such as UPDATE, DELETE, or INSERT to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view non-updatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions (SUM(), MIN(), MAX(), COUNT(), and so forth)
- DISTINCT
- GROUP BY
- HAVING
- UNION or UNION ALL
- Subquery in the select list
- Join
- Non-updatable view in the FROM clause
- A subquery in the WHERE clause that refers to a table in the FROM clause
- Refers only to literal values (in this case, there is no underlying table to update)
- ALGORITHM = TEMPTABLE (use of a temporary table always makes a view

non-updatable)

With respect to insertability (being updatable with `INSERT` statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references and not derived columns. A derived column is one that is not a simple column reference but is derived from an expression. These are examples of derived columns:

```
3.14159  
col1 + 3  
UPPER(col2)  
col3 / col4  
(subquery)
```

A view that has a mix of simple column references and derived columns is not insertable, but it can be updatable if you update only those columns that are not derived. Consider this view:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because `col2` is derived from an expression. But it is updatable if the update does not try to update `col2`. This update is allowable:

```
UPDATE v SET col1 = 0;
```

This update is not allowable because it attempts to update a derived column:

```
UPDATE v SET col2 = 0;
```

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the `MERGE` algorithm. For this to work, the view must use an inner join (not an outer join or a `UNION`). Also, only a single table in the view definition can be updated, so the `SET` clause must name only columns from one of the tables in the view. Views that use `UNION ALL` are disallowed even though they might be theoretically updatable, because the implementation uses

temporary tables to process them.

For a multiple-table updatable view, INSERT can work if it inserts into a single table. DELETE is not supported.

The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts or updates to rows except those for which the WHERE clause in the *select_statement* is true.

In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. The LOCAL keyword restricts the CHECK OPTION only to the view being defined. CASCADED causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is CASCADED. Consider the definitions for the following table and set of views:

```
mysql> CREATE TABLE t1 (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
-> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
-> WITH LOCAL CHECK OPTION;
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
-> WITH CASCADED CHECK OPTION;
```

Here the v2 and v3 views are defined in terms of another view, v1. v2 has a LOCAL check option, so inserts are tested only against the v2 check. v3 has a CASCADED check option, so inserts are tested not only against its own check, but against those of underlying views. The following statements illustrate these differences:

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

The updatability of views may be affected by the value of the `updatable_views_with_limit` system variable. See [Section 5.2.2, “Server System Variables”](#).

The CREATE VIEW statement was added in MySQL 5.0.1. The WITH CHECK OPTION clause was implemented in MySQL 5.0.2.

19.3. DROP VIEW Syntax

```
DROP VIEW [IF EXISTS]
  view_name [, view_name] ...
  [RESTRICT | CASCADE]
```

`DROP VIEW` removes one or more views. You must have the `DROP` privilege for each view. If any of the views named in the argument list do not exist, MySQL returns an error indicating by name which non-existing views it was unable to drop, but it also drops all of the views in the list that do exist.

The `IF EXISTS` clause prevents an error from occurring for views that don't exist. When this clause is given, a `NOTE` is generated for each non-existent view. See [Section 13.5.4.25, “SHOW WARNINGS Syntax”](#).

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

This statement was added in MySQL 5.0.1.

Chapter 20. The INFORMATION_SCHEMA Database

Table of Contents

- [20.1. The INFORMATION_SCHEMA SCHEMATA Table](#)
- [20.2. The INFORMATION_SCHEMA TABLES Table](#)
- [20.3. The INFORMATION_SCHEMA COLUMNS Table](#)
- [20.4. The INFORMATION_SCHEMA STATISTICS Table](#)
- [20.5. The INFORMATION_SCHEMA USER PRIVILEGES Table](#)
- [20.6. The INFORMATION_SCHEMA SCHEMA PRIVILEGES Table](#)
- [20.7. The INFORMATION_SCHEMA TABLE PRIVILEGES Table](#)
- [20.8. The INFORMATION_SCHEMA COLUMN PRIVILEGES Table](#)
- [20.9. The INFORMATION_SCHEMA CHARACTER SETS Table](#)
- [20.10. The INFORMATION_SCHEMA COLLATIONS Table](#)
- [20.11. The INFORMATION_SCHEMA COLLATION CHARACTER SET APPLICABILITY Table](#)
- [20.12. The INFORMATION_SCHEMA TABLE CONSTRAINTS Table](#)
- [20.13. The INFORMATION_SCHEMA KEY COLUMN USAGE Table](#)
- [20.14. The INFORMATION_SCHEMA ROUTINES Table](#)
- [20.15. The INFORMATION_SCHEMA VIEWS Table](#)
- [20.16. The INFORMATION_SCHEMA TRIGGERS Table](#)
- [20.17. Other INFORMATION_SCHEMA Tables](#)
- [20.18. Extensions to SHOW Statements](#)

INFORMATION_SCHEMA provides access to database metadata.

Metadata is data about the data, such as the name of a database or table, the data type of a column, or access privileges. Other terms that sometimes are used for this information are *data dictionary* and *system catalog*.

INFORMATION_SCHEMA is the information database, the place that stores information about all the other databases that the MySQL server maintains. Inside INFORMATION_SCHEMA there are several read-only tables. They are actually views, not base tables, so there are no files associated with them.

In effect, we have a database named INFORMATION_SCHEMA, although the server does not create a database directory with that name. It is possible to select INFORMATION_SCHEMA as the default database with a USE statement, but it is

possible only to read the contents of tables. You cannot insert into them, update them, or delete from them.

Here is an example of a statement that retrieves information from INFORMATION_SCHEMA:

```
mysql> SELECT table_name, table_type, engine
-> FROM information_schema.tables
-> WHERE table_schema = 'db5'
-> ORDER BY table_name DESC;
```

```
+-----+-----+-----+
| table_name | table_type | engine |
+-----+-----+-----+
| v56       | VIEW      | NULL  |
| v3        | VIEW      | NULL  |
| v2        | VIEW      | NULL  |
| v         | VIEW      | NULL  |
| tables    | BASE TABLE | MyISAM |
| t7        | BASE TABLE | MyISAM |
| t3        | BASE TABLE | MyISAM |
| t2        | BASE TABLE | MyISAM |
| t         | BASE TABLE | MyISAM |
| pk        | BASE TABLE | InnoDB |
| loop      | BASE TABLE | MyISAM |
| kurs      | BASE TABLE | MyISAM |
| k         | BASE TABLE | MyISAM |
| into      | BASE TABLE | MyISAM |
| goto      | BASE TABLE | MyISAM |
| fk2       | BASE TABLE | InnoDB |
| fk        | BASE TABLE | InnoDB |
+-----+-----+-----+
17 rows in set (0.01 sec)
```

Explanation: The statement requests a list of all the tables in database db5, in reverse alphabetical order, showing just three pieces of information: the name of the table, its type, and its storage engine.

Each MySQL user has the right to access these tables, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges.

The `SELECT ... FROM INFORMATION_SCHEMA` statement is intended as a more consistent way to provide access to the information provided by the various `SHOW` statements that MySQL supports (`SHOW DATABASES`, `SHOW TABLES`, and so forth). Using `SELECT` has these advantages, compared to `SHOW`:

- It conforms to Codd's rules. That is, all access is done on tables.
- Nobody needs to learn a new statement syntax. Because they already know how `SELECT` works, they only need to learn the object names.
- The implementor need not worry about adding keywords.
- There are millions of possible output variations, instead of just one. This provides more flexibility for applications that have varying requirements about what metadata they need.
- Migration is easier because every other DBMS does it this way.

However, because `SHOW` is popular with MySQL employees and users, and because it might be confusing were it to disappear, the advantages of conventional syntax are not a sufficient reason to eliminate `SHOW`. In fact, along with the implementation of `INFORMATION_SCHEMA`, there are enhancements to `SHOW` as well. These are described in [Section 20.18, “Extensions to SHOW Statements”](#).

There is no difference between the privileges required for `SHOW` statements and those required to select information from `INFORMATION_SCHEMA`. In either case, you have to have some privilege on an object in order to see information about it.

The implementation for the `INFORMATION_SCHEMA` table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such column is the `ENGINE` column in the `INFORMATION_SCHEMA.TABLES` table.

Although other DBMSs use a variety of names, like `syscat` or `system`, the standard name is `INFORMATION_SCHEMA`.

The following sections describe each of the tables and columns that are in `INFORMATION_SCHEMA`. For each column, there are three pieces of information:

- “INFORMATION_SCHEMA Name” indicates the name for the column in the INFORMATION_SCHEMA table. This corresponds to the standard SQL name unless the “Remarks” field says “MySQL extension.”
- “SHOW Name” indicates the equivalent field name in the closest SHOW statement, if there is one.
- “Remarks” provides additional information where applicable. If this field is NULL, it means that the value of the column is always NULL. If this field says “MySQL extension,” the column is a MySQL extension to standard SQL.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked “MySQL extension”. (For example, we changed COLLATION to TABLE_COLLATION in the TABLES table.) See the list of reserved words near the end of this article:

<http://www.dbazine.com/gulutzan5.shtml>.

The definition for character columns (for example, TABLES.TABLE_NAME) is generally VARCHAR(N) CHARACTER SET utf8 where *N* is at least 64.

Each section indicates what SHOW statement is equivalent to a SELECT that retrieves information from INFORMATION_SCHEMA, if there is such a statement.

Note: At present, there are some missing columns and some columns out of order. We are working on this and update the documentation as changes are made.

20.1. The INFORMATION_SCHEMA SCHEMATA Table

A schema is a database, so the SCHEMATA table provides information about databases.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CATALOG_NAME		NULL
SCHEMA_NAME		Database
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		NULL

Notes:

- DEFAULT_COLLATION_NAME was added in MySQL 5.0.6.

The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`  
FROM INFORMATION_SCHEMA.SCHEMATA  
[WHERE SCHEMA_NAME LIKE 'wild']
```

```
SHOW DATABASES  
[LIKE 'wild']
```

20.2. The INFORMATION_SCHEMA TABLES Table

The TABLES table provides information about tables in databases.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA	Table_...	
TABLE_NAME	Table_...	
TABLE_TYPE		
ENGINE	Engine	MySQL extension
VERSION	Version	MySQL extension
ROW_FORMAT	Row_format	MySQL extension
TABLE_ROWS	Rows	MySQL extension
AVG_ROW_LENGTH	Avg_row_length	MySQL extension
DATA_LENGTH	Data_length	MySQL extension
MAX_DATA_LENGTH	Max_data_length	MySQL extension
INDEX_LENGTH	Index_length	MySQL extension
DATA_FREE	Data_free	MySQL extension
AUTO_INCREMENT	Auto_increment	MySQL extension
CREATE_TIME	Create_time	MySQL extension
UPDATE_TIME	Update_time	MySQL extension
CHECK_TIME	Check_time	MySQL extension
TABLE_COLLATION	Collation	MySQL extension
CHECKSUM	Checksum	MySQL extension
CREATE_OPTIONS	Create_options	MySQL extension
TABLE_COMMENT	Comment	MySQL extension

Notes:

- TABLE_SCHEMA and TABLE_NAME are a single field in a SHOW display, for example Table_in_db1.

- TABLE_TYPE should be BASE TABLE or VIEW. If table is temporary, then TABLE_TYPE = TEMPORARY. (There are no temporary views, so this is not ambiguous.)
- The TABLE_ROWS column is NULL if the table is in the INFORMATION_SCHEMA database. For InnoDB tables, the row count is only a rough estimate used in SQL optimization.
- We have nothing for the table's default character set. TABLE_COLLATION is close, because collation names begin with a character set name.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
  [WHERE table_schema = 'db_name']
  [WHERE|AND table_name LIKE 'wild']
```

```
SHOW TABLES
  [FROM db_name]
  [LIKE 'wild']
```

20.3. The INFORMATION_SCHEMA COLUMNS Table

The COLUMNS table provides information about columns in tables.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		see notes
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE	Type	MySQL extension
COLUMN_KEY	Key	MySQL extension
EXTRA	Extra	MySQL extension
COLUMN_COMMENT	Comment	MySQL extension

Notes:

- In SHOW, the Type display includes values from several different COLUMNS columns.
- ORDINAL_POSITION is necessary because you might want to say ORDER BY ORDINAL_POSITION. Unlike SHOW, SELECT does not have automatic ordering.
- CHARACTER_OCTET_LENGTH should be the same as

CHARACTER_MAXIMUM_LENGTH, except for multi-byte character sets.

- CHARACTER_SET_NAME can be derived from collation. For example, if you say `SHOW FULL COLUMNS FROM t`, and you see in the `Collation` column a value of `latin1_swedish_ci`, the character set is what's before the first underscore: `latin1`.

The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'tbl_name'
[AND table_schema = 'db_name']
[AND column_name LIKE 'wild']
```

```
SHOW COLUMNS
FROM tbl_name
[FROM db_name]
[LIKE 'wild']
```

20.4. The INFORMATION_SCHEMA STATISTICS Table

The STATISTICS table provides information about table indexes.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		= Database
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension

Notes:

- There is no standard table for indexes. The preceding list is similar to what SQL Server 2000 returns for `sp_statistics`, except that we replaced the name `QUALIFIER` with `CATALOG` and we replaced the name `OWNER` with `SCHEMA`.

Clearly, the preceding table and the output from `SHOW INDEX` are derived from the same parent. So the correlation is already close.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
```

```
WHERE table_name = 'tbl_name'  
[AND table_schema = 'db_name']
```

```
SHOW INDEX  
FROM tbl_name  
[FROM db_name]
```

20.5. The INFORMATION_SCHEMA USER_PRIVILEGES Table

The USER_PRIVILEGES table provides information about global privileges. This information comes from the `mysql.user` grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		'user_name'@'host_name' value, MySQL extension
TABLE_CATALOG		NULL, MySQL extension
PRIVILEGE_TYPE		MySQL extension
IS_GRANTABLE		MySQL extension

Notes:

- This is a non-standard table. It takes its values from the `mysql.user` table.

20.6. The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table

The SCHEMA_PRIVILEGES table provides information about schema (database) privileges. This information comes from the `mysql.db` grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		'user_name'@'host_name' value, MySQL extension
TABLE_CATALOG		NULL, MySQL extension
TABLE_SCHEMA		MySQL extension
PRIVILEGE_TYPE		MySQL extension
IS_GRANTABLE		MySQL extension

Notes:

- This is a non-standard table. It takes its values from the `mysql.db` table.

20.7. The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The TABLE_PRIVILEGES table provides information about table privileges. This information comes from the `mysql.tables_priv` grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		'user_name'@'host_name' value
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP, CREATE VIEW.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
```

```
SHOW GRANTS ...
```

20.8. The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table

The COLUMN_PRIVILEGES table provides information about column privileges. This information comes from the `mysql.columns_priv` grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		'user_name'@'host_name' value
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- In the output from `SHOW FULL COLUMNS`, the privileges are all in one field and in lowercase, for example, `select, insert, update, references`. In `COLUMN_PRIVILEGES`, there is one privilege per row, in uppercase.
- `PRIVILEGE_TYPE` can contain one (and only one) of these values: `SELECT, INSERT, UPDATE, REFERENCES`.
- If the user has `GRANT OPTION` privilege, `IS_GRANTABLE` should be `YES`. Otherwise, `IS_GRANTABLE` should be `NO`. The output does not list `GRANT OPTION` as a separate privilege.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
```

```
SHOW GRANTS ...
```

20.9. The INFORMATION_SCHEMA CHARACTER_SETS Table

The CHARACTER_SETS table provides information about available character sets.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CHARACTER_SET_NAME	Charset	
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPTION	Description	MySQL extension
MAXLEN	Maxlen	MySQL extension

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS  
  [WHERE name LIKE 'wild']
```

```
SHOW CHARACTER SET  
  [LIKE 'wild']
```

20.10. The INFORMATION_SCHEMA COLLATIONS Table

The COLLATIONS table provides information about collations for each character set.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	MySQL extension
ID	Id	MySQL extension
IS_DEFAULT	Default	MySQL extension
IS_COMPILED	Compiled	MySQL extension
SORTLEN	Sortlen	MySQL extension

The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS  
  [WHERE collation_name LIKE 'wild']
```

```
SHOW COLLATION  
  [LIKE 'wild']
```

20.11. The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The COLLATION_CHARACTER_SET_APPLICABILITY table indicates what character set is applicable for what collation. The columns are equivalent to the first two display fields that we get from SHOW COLLATION.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	

20.12. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The TABLE_CONSTRAINTS table describes which tables have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

Notes:

- The CONSTRAINT_TYPE value can be UNIQUE, PRIMARY KEY, or FOREIGN KEY.
- The UNIQUE and PRIMARY KEY information is about the same as what you get from the key_name field in the output from SHOW INDEX when the Non_unique field is 0.
- The CONSTRAINT_TYPE column can contain one of these values: UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. This is a CHAR (not ENUM) column. The CHECK value is not available until we support CHECK.

20.13. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The KEY_COLUMN_USAGE table describes which key columns have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSITION_IN_UNIQUE_CONSTRAINT		
REFERENCED_TABLE_SCHEMA		
REFERENCED_TABLE_NAME		
REFERENCED_COLUMN_NAME		

Notes:

- If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
- The value of ORDINAL_POSITION is the column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.
- The value of POSITION_IN_UNIQUE_CONSTRAINT is NULL for unique and primary-key constraints. For foreign-key constraints, it is the ordinal position in key of the table that is being referenced.

For example, suppose that there are two tables name t1 and t3 that have the following definitions:

```
CREATE TABLE t1
(
  s1 INT,
  s2 INT,
  s3 INT,
  PRIMARY KEY(s3)
) ENGINE=InnoDB;
```

```
CREATE TABLE t3
(
  s1 INT,
  s2 INT,
  s3 INT,
  KEY(s1),
  CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

For those two tables, the KEY_COLUMN_USAGE table has two rows:

- One row with CONSTRAINT_NAME = 'PRIMARY', TABLE_NAME = 't1', COLUMN_NAME = 's3', ORDINAL_POSITION = 1, POSITION_IN_UNIQUE_CONSTRAINT = NULL.
 - One row with CONSTRAINT_NAME = 'CO', TABLE_NAME = 't3', COLUMN_NAME = 's2', ORDINAL_POSITION = 1, POSITION_IN_UNIQUE_CONSTRAINT = 1.
- REFERENCED_TABLE_SCHEMA, REFERENCED_TABLE_NAME, and REFERENCED_COLUMN_NAME were added in MySQL 5.0.6.

20.14. The INFORMATION_SCHEMA ROUTINES Table

The ROUTINES table provides information about stored routines (both procedures and functions). The ROUTINES table does not include user-defined functions (UDFs) at this time.

The column named “mysql.proc name” indicates the mysql.proc table column that corresponds to the INFORMATION_SCHEMA.ROUTINES table column, if any.

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
SPECIFIC_NAME	specific_name	
ROUTINE_CATALOG		NULL
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	{PROCEDURE FUNCTION}
DTD_IDENTIFIER		(data type descriptor)
ROUTINE_BODY		SQL
ROUTINE_DEFINITION	body	
EXTERNAL_NAME		NULL
EXTERNAL_LANGUAGE	language	NULL
PARAMETER_STYLE		SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
SQL_PATH		NULL
SECURITY_TYPE	security_type	
CREATED	created	
LAST_ALTERED	modified	
SQL_MODE	sql_mode	MySQL extension
ROUTINE_COMMENT	comment	MySQL extension
DEFINER	definer	MySQL extension

Notes:

- MySQL calculates EXTERNAL_LANGUAGE thus:
 - If `mysql.proc.language='SQL'`, EXTERNAL_LANGUAGE is NULL
 - Otherwise, EXTERNAL_LANGUAGE is what is in `mysql.proc.language`. However, we do not have external languages yet, so it is always NULL.

20.15. The INFORMATION_SCHEMA VIEWS Table

The VIEWS table provides information about views in databases. You must have the SHOW VIEW privilege to access this table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
DEFINER		
SECURITY_TYPE		

Notes:

- The VIEW_DEFINITION column has most of what you see in the Create Table field that SHOW CREATE VIEW produces. Skip the words before SELECT and skip the words WITH CHECK OPTION. Suppose that the original statement was:

```
CREATE VIEW v AS
  SELECT s2,s1 FROM t
  WHERE s1 > 5
  ORDER BY s1
  WITH CHECK OPTION;
```

Then the view definition looks like this:

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- The CHECK_OPTION column always has a value of NONE.
- The IS_UPDATABLE column is YES if the view is updatable, NO if the view is not updatable.

- The `DEFINER` and `SECURITY_TYPE` columns were added in MySQL 5.0.14. `DEFINER` indicates who defined the view. `SECURITY_TYPE` has a value of `DEFINER` or `INVOKER`.

20.16. The INFORMATION_SCHEMA TRIGGERS Table

The TRIGGERS table provides information about triggers. You must have the SUPER privilege to access this table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TRIGGER_CATALOG		NULL
TRIGGER_SCHEMA		
TRIGGER_NAME	Trigger	
EVENT_MANIPULATION	Event	
EVENT_OBJECT_CATALOG		NULL
EVENT_OBJECT_SCHEMA		
EVENT_OBJECT_TABLE	Table	
ACTION_ORDER		0
ACTION_CONDITION		NULL
ACTION_STATEMENT	Statement	
ACTION_ORIENTATION		ROW
ACTION_TIMING	Timing	
ACTION_REFERENCE_OLD_TABLE		NULL
ACTION_REFERENCE_NEW_TABLE		NULL
ACTION_REFERENCE_OLD_ROW		OLD
ACTION_REFERENCE_NEW_ROW		NEW
CREATED		NULL (0)
SQL_MODE		MySQL extension
DEFINER		MySQL extension

Notes:

- The TRIGGERS table was added in MySQL 5.0.10.
- The TRIGGER_SCHEMA and TRIGGER_NAME columns contain the name of the database in which the trigger occurs and the trigger name, respectively.

- The `EVENT_MANIPULATION` column contains one of the values `'INSERT'`, `'DELETE'`, or `'UPDATE'`.
- As noted in [Chapter 18, Triggers](#), every trigger is associated with exactly one table. The `EVENT_OBJECT_SCHEMA` and `EVENT_OBJECT_TABLE` columns contain the database in which this table occurs, and the table's name.
- The `ACTION_ORDER` statement contains the ordinal position of the trigger's action within the list of all similar triggers on the same table. Currently, this value is always 0, because it is not possible to have more than one trigger with the same `EVENT_MANIPULATION` and `ACTION_TIMING` on the same table.
- The `ACTION_STATEMENT` column contains the statement to be executed when the trigger is invoked. This is the same as the text displayed in the `Statement` column of the output from `SHOW TRIGGERS`. Note that this text uses UTF-8 encoding.
- The `ACTION_ORIENTATION` column always contains the value `'ROW'`.
- The `ACTION_TIMING` column contains one of the two values `'BEFORE'` or `'AFTER'`.
- The columns `ACTION_REFERENCE_OLD_ROW` and `ACTION_REFERENCE_NEW_ROW` contain the old and new column identifiers, respectively. This means that `ACTION_REFERENCE_OLD_ROW` always contains the value `'OLD'` and `ACTION_REFERENCE_NEW_ROW` always contains the value `'NEW'`.
- The `SQL_MODE` column shows the server SQL mode that was in effect at the time when the trigger was created (and thus which remains in effect for this trigger whenever it is invoked, *regardless of the current server SQL mode*). The possible range of values for this column is the same as that of the `sql_mode` system variable. See [Section 5.2.5, “The Server SQL Mode”](#).
- The `DEFINER` column was added in MySQL 5.0.17. `DEFINER` indicates who defined the trigger.
- The following columns currently always contain `NULL`: `TRIGGER_CATALOG`, `EVENT_OBJECT_CATALOG`, `ACTION_CONDITION`, `ACTION_REFERENCE_OLD_TABLE`, `ACTION_REFERENCE_NEW_TABLE`, and `CREATED`.

Example, using the ins_sum trigger defined in [Section 18.3, “Using Triggers”](#):

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS\G
***** 1. row *****
      TRIGGER_CATALOG: NULL
      TRIGGER_SCHEMA: test
      TRIGGER_NAME: ins_sum
      EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: NULL
      EVENT_OBJECT_SCHEMA: test
      EVENT_OBJECT_TABLE: account
      ACTION_ORDER: 0
      ACTION_CONDITION: NULL
      ACTION_STATEMENT: SET @sum = @sum + NEW.amount
      ACTION_ORIENTATION: ROW
      ACTION_TIMING: BEFORE
      ACTION_REFERENCE_OLD_TABLE: NULL
      ACTION_REFERENCE_NEW_TABLE: NULL
      ACTION_REFERENCE_OLD_ROW: OLD
      ACTION_REFERENCE_NEW_ROW: NEW
      CREATED: NULL
      SQL_MODE:
      DEFINER: me@localhost
```

See also [Section 13.5.4.23, “SHOW TRIGGERS Syntax”](#).

20.17. Other INFORMATION_SCHEMA Tables

We intend to implement additional INFORMATION_SCHEMA tables. In particular, we acknowledge the need for the PARAMETERS and REFERENTIAL_CONSTRAINTS tables. (REFERENTIAL_CONSTRAINTS is implemented in MySQL 5.1.)

20.18. Extensions to SHOW Statements

Some extensions to SHOW statements accompany the implementation of INFORMATION_SCHEMA:

- SHOW can be used to get information about the structure of INFORMATION_SCHEMA itself.
- Several SHOW statements accept a WHERE clause that provides more flexibility in specifying which rows to display.

These extensions are available beginning with MySQL 5.0.3.

INFORMATION_SCHEMA is an information database, so its name is included in the output from SHOW DATABASES. Similarly, SHOW TABLES can be used with INFORMATION_SCHEMA to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+-----+
| Tables_in_information_schema |
+-----+
| CHARACTER_SETS              |
| COLLATIONS                   |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                     |
| COLUMN_PRIVILEGES           |
| KEY_COLUMN_USAGE            |
| ROUTINES                     |
| SCHEMATA                     |
| SCHEMA_PRIVILEGES           |
| STATISTICS                   |
| TABLES                      |
| TABLE_CONSTRAINTS          |
| TABLE_PRIVILEGES           |
| TRIGGERS                     |
| USER_PRIVILEGES             |
| VIEWS                        |
+-----+
16 rows in set (0.00 sec)
```

SHOW COLUMNS and DESCRIBE can display information about the columns in individual INFORMATION_SCHEMA tables.

Several SHOW statement have been extended to allow a WHERE clause:

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW KEYS
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW VARIABLES
```

The WHERE clause, if present, is evaluated against the column names displayed by the SHOW statement. For example, the SHOW CHARACTER SET statement produces these output columns:

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation	Max
big5	Big5 Traditional Chinese	big5_chinese_ci	
dec8	DEC West European	dec8_swedish_ci	
cp850	DOS West European	cp850_general_ci	
hp8	HP West European	hp8_english_ci	
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	
latin1	cp1252 West European	latin1_swedish_ci	
latin2	ISO 8859-2 Central European	latin2_general_ci	
...			

To use a WHERE clause with SHOW CHARACTER SET, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string 'japanese':

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%'
```

Charset	Description	Default collation	Maxlen
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

This statement displays the multi-byte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
euckr	EUC-KR Korean	euckr_korean_ci	2
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpm	UJIS for Windows Japanese	eucjpm_japanese_ci	3

Chapter 21. Precision Math

Table of Contents

[21.1. Types of Numeric Values](#)

[21.2. DECIMAL Data Type Changes](#)

[21.3. Expression Handling](#)

[21.4. Rounding Behavior](#)

[21.5. Precision Math Examples](#)

MySQL 5.0 introduces precision math: numeric value handling that results in more accurate results and more control over invalid values than in earlier versions of MySQL. Precision math is based on two implementation changes:

- The introduction of SQL modes in MySQL 5.0 that control how strict the server is about accepting or rejecting invalid data.
- The introduction in MySQL 5.0.3 of a library for fixed-point arithmetic.

These changes have several implications for numeric operations:

- **More precise calculations:** For exact-value numbers, calculations do not introduce floating-point errors. Instead, exact precision is used. For example, a number such as .0001 is treated as an exact value rather than as an approximation, and summing it 10,000 times produces a result of exactly 1, not a value that merely “close” to 1.
- **Well-defined rounding behavior:** For exact-value numbers, the result of `ROUND()` depends on its argument, not on environmental factors such as how the underlying C library works.
- **Improved platform independence:** Operations on exact numeric values are the same across different platforms such as Windows and Unix.
- **Control over handling of invalid values:** Overflow and division by zero are detectable and can be treated as errors. For example, you can treat a value that is too large for a column as an error rather than having the value truncated to lie within the range of the column's data type. Similarly, you

can treat division by zero as an error rather than as an operation that produces a result of NULL. The choice of which approach to take is determined by the setting of the `sql_mode` system variable.

An important result of these changes is that MySQL provides improved compliance with standard SQL.

The following discussion covers several aspects of how precision math works (including possible incompatibilities with older applications). At the end, some examples are given that demonstrate how MySQL 5.0 handles numeric operations precisely. For information about using the `sql_mode` system variable to control the SQL mode, see [Section 5.2.5, “The Server SQL Mode”](#).

21.1. Types of Numeric Values

The scope of precision math for exact-value operations includes the exact-value data types (`DECIMAL` and integer types) and exact-value numeric literals.

Approximate-value data types and numeric literals still are handled as floating-point numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar need not be both exact-value or both approximate-value. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

21.2. DECIMAL Data Type Changes

This section discusses the characteristics of the `DECIMAL` data type (and its synonyms) as of MySQL 5.0.3, with particular regard to the following topics:

- Maximum number of digits
- Storage format
- Storage requirements
- The non-standard MySQL extension to the upper range of `DECIMAL` columns

Some of these changes result in possible incompatibilities for applications that are written for older versions of MySQL. These incompatibilities are noted throughout this section.

The declaration syntax for a `DECIMAL` column remains `DECIMAL(M,D)`, although the range of values for the arguments has changed somewhat:

- M is the maximum number of digits (the precision). It has a range of 1 to 65. This introduces a possible incompatibility for older applications, because previous versions of MySQL allow a range of 1 to 254.

The precision of 65 digits actually applies as of MySQL 5.0.6. From 5.0.3 to 5.0.5, the precision is 64 digits.

- D is the number of digits to the right of the decimal point (the scale). It has a range of 0 to 30 and must be no larger than M .

The maximum value of 65 for M means that calculations on `DECIMAL` values are accurate up to 65 digits. This limit of 65 digits of precision also applies to exact-value numeric literals, so the maximum range of such literals is different from before. (Prior to MySQL 5.0.3, decimal values could have up to 254 digits. However, calculations were done using floating-point and thus were approximate, not exact.) This change in the range of literal values is another possible source of incompatibility for older applications.

Values for `DECIMAL` columns no longer are represented as strings that require one

byte per digit or sign character. Instead, a binary format is used that packs nine decimal digits into four bytes. This change to `DECIMAL` storage format changes the storage requirements as well. The storage requirements for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and any digits left over require some fraction of four bytes. For example, a `DECIMAL(18, 9)` column has nine digits on either side of the decimal point, so the integer part and the fractional part each require four bytes. A `DECIMAL(20, 10)` column has ten digits on either side of the decimal point. Each part requires four bytes for nine of the digits, and one byte for the remaining digit.

The storage required for leftover digits is given by the following table:

Leftover Digits	Number of Bytes
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	4

As a result of the change from string to numeric format for `DECIMAL` storage, `DECIMAL` columns no longer store a leading `+` character or leading `0` digits. Before MySQL 5.0.3, if you inserted `+0003.1` into a `DECIMAL(5, 1)` column, it was stored as `+0003.1`. As of MySQL 5.0.3, it is stored as `3.1`. Applications that rely on the older behavior must be modified to account for this change.

The change of storage format also means that `DECIMAL` columns no longer support the non-standard extension that allowed values larger than the range implied by the column definition. Formerly, one byte was allocated for storing the sign character. For positive values that needed no sign byte, MySQL allowed an extra digit to be stored instead. For example, a `DECIMAL(3, 0)` column must

support a range of at least -999 to 999, but MySQL would allow storing values from 1000 to 9999 as well, by using the sign byte to store an extra digit. This extension to the upper range of `DECIMAL` columns no longer is allowed. In MySQL 5.0.3 and up, a `DECIMAL(M,D)` column allows at most $M-D$ digits to the left of the decimal point. This can result in an incompatibility if an application has a reliance on MySQL allowing “too-large” values.

The SQL standard requires that the precision of `NUMERIC(M,D)` be *exactly* M digits. For `DECIMAL(M,D)`, the standard requires a precision of at least M digits but allows more. In MySQL, `DECIMAL(M,D)` and `NUMERIC(M,D)` are the same, and both have a precision of exactly M digits.

Summary of incompatibilities:

The following list summarizes the incompatibilities that result from changes to `DECIMAL` column and value handling. You can use it as guide when porting older applications for use with MySQL 5.0.3 and up.

- For `DECIMAL(M,D)`, the maximum M is 65, not 254.
- Calculations involving exact-value decimal numbers are accurate to 65 digits. This is fewer than the maximum number of digits allowed before MySQL 5.0.3 (254 digits), but the exact-value precision is greater. Calculations formerly were done with double-precision floating-point, which has a precision of 52 bits (about 15 decimal digits).
- The non-standard MySQL extension to the upper range of `DECIMAL` columns no longer is supported.
- Leading ‘+’ and ‘0’ characters are not stored.

The behavior used by the server for `DECIMAL` columns in a table depends on the version of MySQL used to create the table. If your server is from MySQL 5.0.3 or higher, but you have `DECIMAL` columns in tables that were created before 5.0.3, the old behavior still applies to those columns. To convert the tables to the newer `DECIMAL` format, dump them with **mysqldump** and reload them.

21.3. Expression Handling

With precision math, exact-value numbers are used as given whenever possible. For example, numbers in comparisons are used exactly as given without a change in value. In strict SQL mode, for `INSERT` into a column with an exact data type (`DECIMAL` or integer), a number is inserted with its exact value if it is within the column range. When retrieved, the value should be the same as what was inserted. (Without strict mode, truncation for `INSERT` is allowable.)

Handling of a numeric expression depends on what kind of values the expression contains:

- If any approximate values are present, the expression is approximate and is evaluated using floating-point arithmetic.
- If no approximate values are present, the expression contains only exact values. If any exact value contains a fractional part (a value following the decimal point), the expression is evaluated using `DECIMAL` exact arithmetic and has a precision of 65 digits. (The term “exact” is subject to the limits of what can be represented in binary. For example, $1.0/3.0$ can be approximated in decimal notation as $.333\dots$, but not written as an exact number, so $(1.0/3.0)*3.0$ does not evaluate to exactly 1.0 .)
- Otherwise, the expression contains only integer values. The expression is exact and is evaluated using integer arithmetic and has a precision the same as `BIGINT` (64 bits).

If a numeric expression contains any strings, they are converted to double-precision floating-point values and the expression is approximate.

Inserts into numeric columns are affected by the SQL mode, which is controlled by the `sql_mode` system variable. (See [Section 5.2.5, “The Server SQL Mode”](#).) The following discussion mentions strict mode (selected by the `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` mode values) and `ERROR_FOR_DIVISION_BY_ZERO`. To turn on all restrictions, you can simply use `TRADITIONAL` mode, which includes both strict mode values and `ERROR_FOR_DIVISION_BY_ZERO`:

```
mysql> SET sql_mode='TRADITIONAL';
```

If a number is inserted into an exact type column (DECIMAL or integer), it is inserted with its exact value if it is within the column range.

If the value has too many digits in the fractional part, rounding occurs and a warning is generated. Rounding is done as described in [Section 21.4, “Rounding Behavior”](#).

If the value has too many digits in the integer part, it is too large and is handled as follows:

- If strict mode is not enabled, the value is truncated to the nearest legal value and a warning is generated.
- If strict mode is enabled, an overflow error occurs.

Underflow is not detected, so underflow handling is undefined.

By default, division by zero produces a result of NULL and no warning. With the `ERROR_FOR_DIVISION_BY_ZERO` SQL mode enabled, MySQL handles division by zero differently:

- If strict mode is not enabled, a warning occurs.
- If strict mode is enabled, inserts and updates involving division by zero are prohibited, and an error occurs.

In other words, inserts and updates involving expressions that perform division by zero can be treated as errors, but this requires `ERROR_FOR_DIVISION_BY_ZERO` in addition to strict mode.

Suppose that we have this statement:

```
INSERT INTO t SET i = 1/0;
```

This is what happens for combinations of strict and `ERROR_FOR_DIVISION_BY_ZERO` modes:

sql_mode Value	Result

' ' (Default)	No warning, no error; i is set to NULL.
strict	No warning, no error; i is set to NULL.
ERROR_FOR_DIVISION_BY_ZERO	Warning, no error; i is set to NULL.
strict,ERROR_FOR_DIVISION_BY_ZERO	Error condition; no row is inserted.

For inserts of strings into numeric columns, conversion from string to number is handled as follows if the string has non-numeric contents:

- A string that does not begin with a number cannot be used as a number and produces an error in strict mode, or a warning otherwise. *This includes the empty string.*
- A string that begins with a number can be converted, but the trailing non-numeric portion is truncated. If the truncated portion contains anything other than spaces, this produces an error in strict mode, or a warning otherwise.

21.4. Rounding Behavior

This section discusses precision math rounding for the `ROUND()` function and for inserts into columns with exact-value types (`DECIMAL` and integer).

The `ROUND()` function rounds differently depending on whether its argument is exact or approximate:

- For exact-value numbers, `ROUND()` uses the “round half up” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the “round to nearest even” rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3          |                2 |
+-----+-----+
```

For inserts into a `DECIMAL` or integer column, the target is an exact data type, so rounding uses “round half up,” regardless of whether the value to be inserted is exact or approximate:

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 2

mysql> SELECT d FROM t;
```

d
3
3

21.5. Precision Math Examples

This section provides some examples that show how precision math improves query results in MySQL 5 compared to older versions.

Example 1. Numbers are used with their exact value as given when possible.

Before MySQL 5.0.3, numbers that are treated as floating-point values produce inexact results:

```
mysql> SELECT .1 + .2 = .3;
+-----+
| .1 + .2 = .3 |
+-----+
|              0 |
+-----+
```

As of MySQL 5.0.3, numbers are used as given when possible:

```
mysql> SELECT .1 + .2 = .3;
+-----+
| .1 + .2 = .3 |
+-----+
|              1 |
+-----+
```

For floating-point values, results are inexact:

```
mysql> SELECT .1E0 + .2E0 = .3E0;
+-----+
| .1E0 + .2E0 = .3E0 |
+-----+
|                      0 |
+-----+
```

Another way to see the difference in exact and approximate value handling is to add a small number to a sum many times. Consider the following stored procedure, which adds .0001 to a variable 1,000 times.

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
```

```

DECLARE f FLOAT DEFAULT 0;
WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
    SET i = i + 1;
END WHILE;
SELECT d, f;
END;

```

The sum for both *d* and *f* logically should be 1, but that is true only for the decimal calculation. The floating-point calculation introduces small errors:

```

+-----+-----+
| d      | f              |
+-----+-----+
| 1.0000 | 0.999999999999 |
+-----+-----+

```

Example 2. Multiplication is performed with the scale required by standard SQL. That is, for two numbers *x1* and *x2* that have scale *s1* and *s2*, the scale of the result is *s1* + *s2*:

Before MySQL 5.0.3, this is what happens:

```

mysql> SELECT .01 * .01;
+-----+
| .01 * .01 |
+-----+
|      0.00 |
+-----+

```

The displayed value is incorrect. The value was calculated correctly in this case, but not displayed to the required scale. To see that the calculated value actually was .0001, try this:

```

mysql> SELECT .01 * .01 + .0000;
+-----+
| .01 * .01 + .0000 |
+-----+
|           0.0001 |
+-----+

```

As of MySQL 5.0.3, the displayed scale is correct:

```

mysql> SELECT .01 * .01;
+-----+

```

```

| .01 * .01 |
+-----+
| 0.0001    |
+-----+

```

Example 3. Rounding behavior is well-defined.

Before MySQL 5.0.3, rounding behavior (for example, with the `ROUND()` function) is dependent on the implementation of the underlying C library. This results in inconsistencies from platform to platform. For example, you might get a different value on Windows than on Linux, or a different value on x86 machines than on PowerPC machines.

As of MySQL 5.0.3, rounding happens like this:

Rounding for exact-value columns (`DECIMAL` and `integer`) and exact-valued numbers uses the “round half up” rule. Values with a fractional part of .5 or greater are rounded away from zero to the nearest integer, as shown here:

```

mysql> SELECT ROUND(2.5), ROUND(-2.5);
+-----+-----+
| ROUND(2.5) | ROUND(-2.5) |
+-----+-----+
| 3          | -3          |
+-----+-----+

```

However, rounding for floating-point values uses the C library, which on many systems uses the “round to nearest even” rule. Values with any fractional part on such systems are rounded to the nearest even integer:

```

mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
+-----+-----+
| ROUND(2.5E0) | ROUND(-2.5E0) |
+-----+-----+
|          2   |          -2   |
+-----+-----+

```

Example 4. In strict mode, inserting a value that is too large results in overflow and causes an error, rather than truncation to a legal value.

Before MySQL 5.0.2 (or in 5.0.2 and later, without strict mode), truncation to a legal value occurs:

```

mysql> CREATE TABLE t (i TINYINT);

```

Query OK, 0 rows affected (0.01 sec)

```
mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)
```

```
mysql> SELECT i FROM t;
+-----+
| i     |
+-----+
| 127  |
+-----+
1 row in set (0.00 sec)
```

As of MySQL 5.0.2, overflow occurs if strict mode is in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at ro
```

```
mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

Example 5: In strict mode and with `ERROR_FOR_DIVISION_BY_ZERO` set, division by zero causes an error, and not a result of `NULL`.

Before MySQL 5.0.2 (or when not using strict mode in 5.0.2 or a later version), division by zero has a result of `NULL`:

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT i FROM t;
+-----+
| i     |
+-----+
| NULL  |
+-----+
1 row in set (0.00 sec)
```

As of MySQL 5.0.2, division by zero is an error if the proper SQL modes are in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0
```

```
mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

Example 6. Prior to MySQL 5.0.3 (before precision math was introduced), exact-value and approximate-value literals both are converted to double-precision floating-point values:

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
| 4.1.18-log |
+-----+
1 row in set (0.01 sec)
```

```
mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.07 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

```
mysql> DESCRIBE t;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| a     | double(3,1)  |      |     | 0.0     |      |
| b     | double       |      |     | 0       |      |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.04 sec)
```

As of MySQL 5.0.3, the approximate-value literal still is converted to floating-point, but the exact-value literal is handled as DECIMAL:

```
mysql> SELECT VERSION();
+-----+
| VERSION() |
+-----+
```

```
| 5.0.19-log |
+-----+
1 row in set (0.17 sec)
```

```
mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.19 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

```
mysql> DESCRIBE t;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| a     | decimal(2,1) unsigned | NO   |     | 0.0     |       |
| b     | double              | NO   |     | 0       |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.02 sec)
```

Example 7. If the argument to an aggregate function is an exact numeric type, the result is also an exact numeric type, with a scale at least that of the argument.

Consider these statements:

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

Result before MySQL 5.0.3 (prior to the introduction of precision math in MySQL):

```
mysql> DESCRIBE y;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| AVG(i) | double(17,4)       | YES  |     | NULL    |       |
| AVG(d) | double(17,4)       | YES  |     | NULL    |       |
| AVG(f) | double             | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

The result is a double no matter the argument type.

Result as of MySQL 5.0.3:

```
mysql> DESCRIBE y;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| AVG(i) | decimal(14,4)      | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

AVG(d)	decimal(14,4)	YES		NULL	
AVG(f)	double	YES		NULL	

The result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type. (From MySQL 5.0.3 to 5.0.6, the first two columns are DECIMAL(64,0).)

Chapter 22. APIs and Libraries

Table of Contents

[22.1. libmysqld, the Embedded MySQL Server Library](#)

[22.2. MySQL C API](#)

[22.2.1. C API Data types](#)

[22.2.2. C API Function Overview](#)

[22.2.3. C API Function Descriptions](#)

[22.2.4. C API Prepared Statements](#)

[22.2.5. C API Prepared Statement Data types](#)

[22.2.6. C API Prepared Statement Function Overview](#)

[22.2.7. C API Prepared Statement Function Descriptions](#)

[22.2.8. C API Prepared statement problems](#)

[22.2.9. C API Handling of Multiple Statement Execution](#)

[22.2.10. C API Handling of Date and Time Values](#)

[22.2.11. C API Threaded Function Descriptions](#)

[22.2.12. C API Embedded Server Function Descriptions](#)

[22.2.13. Common Questions and Problems When Using the C API](#)

[22.2.14. Building Client Programs](#)

[22.2.15. How to Make a Threaded Client](#)

[22.3. MySQL PHP API](#)

[22.3.1. Common Problems with MySQL and PHP](#)

[22.3.2. Enabling Both `mysql` and `mysql_i` in PHP](#)

[22.4. MySQL Perl API](#)

[22.5. MySQL C++ API](#)

[22.5.1. Borland C++](#)

[22.6. MySQL Python API](#)

[22.7. MySQL Tcl API](#)

[22.8. MySQL Eiffel Wrapper](#)

[22.9. MySQL Program Development Utilities](#)

[22.9.1. `mysql2mysql` — Convert mSQL Programs for Use with MySQL](#)

[22.9.2. `mysql_config` — Get Compile Options for Compiling Clients](#)

This chapter describes the APIs available for MySQL, where to get them, and how to use them. The C API is the most extensively covered, because it was developed by the MySQL team, and is the basis for most of the other APIs. This

chapter also covers some programs that are useful for application developers.

22.1. libmysqld, the Embedded MySQL Server Library

The embedded MySQL server library is NOT part of MySQL 5.0. It is part of previous editions and will be included in future versions, starting with MySQL 5.1. You can find appropriate documentation in the corresponding manuals for these versions. In this manual, only an overview of the embedded library is provided.

The embedded MySQL server library makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and more simple management for embedded applications.

The embedded server library is based on the client/server version of MySQL, which is written in C/C++. Consequently, the embedded server also is written in C/C++. There is no embedded server available in other languages.

The API is identical for the embedded MySQL version and the client/server version. To change an old threaded application to use the embedded library, you normally only have to add calls to the following functions:

Function	When to Call
<code>mysql_server_init()</code>	Should be called before any other MySQL function is called, preferably early in the <code>main()</code> function.
<code>mysql_server_end()</code>	Should be called before your program exits.
<code>mysql_thread_init()</code>	Should be called in each thread you create that accesses MySQL.
<code>mysql_thread_end()</code>	Should be called before calling <code>pthread_exit()</code>

Then you must link your code with `libmysqld.a` instead of `libmysqlclient.a`.

The `mysql_server_xxx()` functions are also included in `libmysqlclient.a` to allow you to change between the embedded and the client/server version by just linking your application with the right library. See [Section 22.2.12.1, “mysql_server_init\(\)”](#).

One difference between the embedded server and the standalone server is that for the embedded server, authentication for connections is disabled by default. To use authentication for the embedded server, specify the `--with-embedded-privilege-control` option when you invoke **configure** to configure your MySQL distribution.

22.2. MySQL C API

The C API code is distributed with MySQL. It is included in the `mysqlclient` library and allows C programs to access a database.

Many of the clients in the MySQL source distribution are written in C. If you are looking for examples that demonstrate how to use the C API, take a look at these clients. You can find these in the `clients` directory in the MySQL source distribution.

Most of the other client APIs (all except Connector/J and Connector/NET) use the `mysqlclient` library to communicate with the MySQL server. This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs, because they are referenced from the library. See [Chapter 8, *Client and Utility Programs*](#), for a list of these variables.

The client has a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (the maximum is 16MB). Because buffer sizes are increased only as demand warrants, simply increasing the default maximum limit does not in itself cause more resources to be used. This size check is mostly a check for erroneous statements and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each thread's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have BLOB values that contain up to 16MB of data, you must have a communication buffer limit of at least 16MB (in both server and client). The client's default maximum is 16MB, but the default maximum in the server is 1MB. You can increase this by changing the value of the `max_allowed_packet` parameter when the server is started. See [Section 7.5.2, “Tuning Server Parameters”](#).

The MySQL server shrinks each communication buffer to `net_buffer_length` bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

For programming with threads, see [Section 22.2.15, “How to Make a Threaded Client”](#). For creating a standalone application which includes the "server" and "client" in the same program (and does not communicate with an external MySQL server), see [Section 22.1, “libmysqld, the Embedded MySQL Server Library”](#).

22.2.1. C API Data types

- `MYSQL`

This structure represents a handle to one database connection. It is used for almost all MySQL functions. You should not try to make a copy of a `MYSQL` structure. There is no guarantee that such a copy will be usable.

- `MYSQL_RES`

This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

- `MYSQL_ROW`

This is a type-safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

- `MYSQL_FIELD`

This structure contains information about a field, such as the field's name, type, and size. Its members are described in more detail here. You may obtain the `MYSQL_FIELD` structures for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

- `MYSQL_FIELD_OFFSET`

This is a type-safe representation of an offset into a MySQL field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row,

beginning at zero.

- `my_ulonglong`

The type used for the number of rows and for `mysql_affected_rows()`, `mysql_num_rows()`, and `mysql_insert_id()`. This type provides a range of 0 to 1.84e19.

On some systems, attempting to print a value of type `my_ulonglong` does not work. To print such a value, convert it to `unsigned long` and use a `%lu` print format. Example:

```
printf ("Number of rows: %lu\n", (unsigned long) mysql_num_rows(
```

The `MYSQL_FIELD` structure contains the members listed here:

- `char * name`

The name of the field, as a null-terminated string. If the field was given an alias with an `AS` clause, the value of `name` is the alias.

- `char * org_name`

The name of the field, as a null-terminated string. Aliases are ignored.

- `char * table`

The name of the table containing this field, if it isn't a calculated field. For calculated fields, the `table` value is an empty string. If the table was given an alias with an `AS` clause, the value of `table` is the alias.

- `char * org_table`

The name of the table, as a null-terminated string. Aliases are ignored.

- `char * db`

The name of the database that the field comes from, as a null-terminated string. If the field is a calculated field, `db` is an empty string.

- `char * catalog`

The catalog name. This value is always "def".

- `char * def`

The default value of this field, as a null-terminated string. This is set only if you use `mysql_list_fields()`.

- `unsigned long length`

The width of the field, as specified in the table definition.

- `unsigned long max_length`

The maximum width of the field for the result set (the length of the longest field value for the rows actually in the result set). If you use `mysql_store_result()` or `mysql_list_fields()`, this contains the maximum length for the field. If you use `mysql_use_result()`, the value of this variable is zero.

- `unsigned int name_length`

The length of name.

- `unsigned int org_name_length`

The length of `org_name`.

- `unsigned int table_length`

The length of table.

- `unsigned int org_table_length`

The length of `org_table`.

- `unsigned int db_length`

The length of db.

- `unsigned int catalog_length`

The length of catalog.

- unsigned int def_length

The length of def.

- unsigned int flags

Different bit-flags for the field. The flags value may have zero or more of the following bits set:

Flag Value	Flag Description
NOT_NULL_FLAG	Field can't be NULL
PRI_KEY_FLAG	Field is part of a primary key
UNIQUE_KEY_FLAG	Field is part of a unique key
MULTIPLE_KEY_FLAG	Field is part of a non-unique key
UNSIGNED_FLAG	Field has the UNSIGNED attribute
ZEROFILL_FLAG	Field has the ZEROFILL attribute
BINARY_FLAG	Field has the BINARY attribute
AUTO_INCREMENT_FLAG	Field has the AUTO_INCREMENT attribute
ENUM_FLAG	Field is an ENUM (deprecated)
SET_FLAG	Field is a SET (deprecated)
BLOB_FLAG	Field is a BLOB or TEXT (deprecated)
TIMESTAMP_FLAG	Field is a TIMESTAMP (deprecated)

Use of the BLOB_FLAG, ENUM_FLAG, SET_FLAG, and TIMESTAMP_FLAG flags is deprecated because they indicate the type of a field rather than an attribute of its type. It is preferable to test `field->type` against `MYSQL_TYPE_BLOB`, `MYSQL_TYPE_ENUM`, `MYSQL_TYPE_SET`, or `MYSQL_TYPE_TIMESTAMP` instead.

The following example illustrates a typical use of the flags value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

You may use the following convenience macros to determine the boolean status of the flags value:

Flag Status	Description
IS_NOT_NULL(flags)	True if this field is defined as NOT NULL
IS_PRI_KEY(flags)	True if this field is a primary key
IS_BLOB(flags)	True if this field is a BLOB or TEXT (deprecated; test field->type instead)

- unsigned int decimals

The number of decimals for numeric fields.

- unsigned int charsetnr

The character set number for the field.

- enum enum_field_types type

The type of the field. The type value may be one of the MYSQL_TYPE_ symbols shown in the following table.

Type Value	Type Description
MYSQL_TYPE_TINY	TINYINT field
MYSQL_TYPE_SHORT	SMALLINT field
MYSQL_TYPE_LONG	INTEGER field
MYSQL_TYPE_INT24	MEDIUMINT field
MYSQL_TYPE_LONGLONG	BIGINT field
MYSQL_TYPE_DECIMAL	DECIMAL or NUMERIC field
MYSQL_TYPE_NEWDECIMAL	Precision math DECIMAL or NUMERIC field (MySQL 5.0.3 and up)
MYSQL_TYPE_FLOAT	FLOAT field
MYSQL_TYPE_DOUBLE	DOUBLE or REAL field
MYSQL_TYPE_BIT	BIT field (MySQL 5.0.3 and up)
MYSQL_TYPE_TIMESTAMP	TIMESTAMP field
MYSQL_TYPE_DATE	DATE field
MYSQL_TYPE_TIME	TIME field

MYSQL_TYPE_DATETIME	DATETIME field
MYSQL_TYPE_YEAR	YEAR field
MYSQL_TYPE_STRING	CHAR or BINARY field
MYSQL_TYPE_VAR_STRING	VARCHAR or VARBINARY field
MYSQL_TYPE_BLOB	BLOB or TEXT field (use max_length to determine the maximum length)
MYSQL_TYPE_SET	SET field
MYSQL_TYPE_ENUM	ENUM field
MYSQL_TYPE_GEOMETRY	Spatial field
MYSQL_TYPE_NULL	NULL-type field
MYSQL_TYPE_CHAR	Deprecated; use MYSQL_TYPE_TINY instead

You can use the `IS_NUM()` macro to test whether a field has a numeric type. Pass the type value to `IS_NUM()` and it evaluates to `TRUE` if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

To distinguish between binary and non-binary data for string data types, check whether the `charsetnr` value is 63. If so, the character set is binary, which indicates binary rather than non-binary data. This is how to distinguish between `BINARY` and `CHAR`, `VARBINARY` and `VARCHAR`, and `BLOB` and `TEXT`.

22.2.2. C API Function Overview

The functions available in the C API are summarized here and described in greater detail in a later section. See [Section 22.2.3, “C API Function Descriptions”](#).

Function	Description
mysql_affected_rows()	Returns the number of rows changed/deleted/inserted by the last <code>UPDATE</code> , <code>DELETE</code> , or <code>INSERT</code> query.
mysql_autocommit()	Toggles autocommit mode on/off.

<u>mysql_change_user()</u>	Changes user and database on an open connection.
<u>mysql_close()</u>	Closes a server connection.
<u>mysql_commit()</u>	Commits the transaction.
<u>mysql_connect()</u>	Connects to a MySQL server. This function is deprecated; use <code>mysql_real_connect()</code> instead.
<u>mysql_create_db()</u>	Creates a database. This function is deprecated; use the SQL statement <code>CREATE DATABASE</code> instead.
<u>mysql_data_seek()</u>	Seeks to an arbitrary row number in a query result set.
<u>mysql_debug()</u>	Does a <code>DEBUG_PUSH</code> with the given string.
<u>mysql_drop_db()</u>	Drops a database. This function is deprecated; use the SQL statement <code>DROP DATABASE</code> instead.
<u>mysql_dump_debug_info()</u>	Makes the server write debug information to the log.
<u>mysql_eof()</u>	Determines whether the last row of a result set has been read. This function is deprecated; <code>mysql_errno()</code> or <code>mysql_error()</code> may be used instead.
<u>mysql_errno()</u>	Returns the error number for the most recently invoked MySQL function.
<u>mysql_error()</u>	Returns the error message for the most recently invoked MySQL function.
<u>mysql_escape_string()</u>	Escapes special characters in a string for use in an SQL statement.
<u>mysql_fetch_field()</u>	Returns the type of the next table field.
<u>mysql_fetch_field_direct()</u>	Returns the type of a table field, given a field number.
<u>mysql_fetch_fields()</u>	Returns an array of all field structures.
<u>mysql_fetch_lengths()</u>	Returns the lengths of all columns in the current row.

<u>mysql_fetch_row()</u>	Fetches the next row from the result set.
<u>mysql_field_seek()</u>	Puts the column cursor on a specified column.
<u>mysql_field_count()</u>	Returns the number of result columns for the most recent statement.
<u>mysql_field_tell()</u>	Returns the position of the field cursor used for the last <code>mysql_fetch_field()</code> .
<u>mysql_free_result()</u>	Frees memory used by a result set.
<u>mysql_get_client_info()</u>	Returns client version information as a string.
<u>mysql_get_client_version()</u>	Returns client version information as an integer.
<u>mysql_get_host_info()</u>	Returns a string describing the connection.
<u>mysql_get_server_version()</u>	Returns version number of server as an integer.
<u>mysql_get_proto_info()</u>	Returns the protocol version used by the connection.
<u>mysql_get_server_info()</u>	Returns the server version number.
<u>mysql_info()</u>	Returns information about the most recently executed query.
<u>mysql_init()</u>	Gets or initializes a MYSQL structure.
<u>mysql_insert_id()</u>	Returns the ID generated for an AUTO_INCREMENT column by the previous query.
<u>mysql_kill()</u>	Kills a given thread.
<u>mysql_library_end()</u>	Finalize MySQL C API library.
<u>mysql_library_init()</u>	Initialize MySQL C API library.
<u>mysql_list_dbs()</u>	Returns database names matching a simple regular expression.
<u>mysql_list_fields()</u>	Returns field names matching a simple regular expression.
<u>mysql_list_processes()</u>	Returns a list of the current server threads.
<u>mysql_list_tables()</u>	Returns table names matching a simple

	regular expression.
<u>mysql_more_results()</u>	Checks whether any more results exist.
<u>mysql_next_result()</u>	Returns/initiates the next result in multiple-statement executions.
<u>mysql_num_fields()</u>	Returns the number of columns in a result set.
<u>mysql_num_rows()</u>	Returns the number of rows in a result set.
<u>mysql_options()</u>	Sets connect options for <code>mysql_connect()</code> .
<u>mysql_ping()</u>	Checks whether the connection to the server is working, reconnecting as necessary.
<u>mysql_query()</u>	Executes an SQL query specified as a null-terminated string.
<u>mysql_real_connect()</u>	Connects to a MySQL server.
<u>mysql_real_escape_string()</u>	Escapes special characters in a string for use in an SQL statement, taking into account the current character set of the connection.
<u>mysql_real_query()</u>	Executes an SQL query specified as a counted string.
<u>mysql_refresh()</u>	Flush or reset tables and caches.
<u>mysql_reload()</u>	Tells the server to reload the grant tables.
<u>mysql_rollback()</u>	Rolls back the transaction.
<u>mysql_row_seek()</u>	Seeks to a row offset in a result set, using value returned from <code>mysql_row_tell()</code> .
<u>mysql_row_tell()</u>	Returns the row cursor position.
<u>mysql_select_db()</u>	Selects a database.
<u>mysql_server_end()</u>	Finalize embedded server library.
<u>mysql_server_init()</u>	Initialize embedded server library.
<u>mysql_set_local_infile_default()</u>	Set the LOAD DATA LOCAL INFILE handler callbacks to their default values.
<u>mysql_set_local_infile_handler()</u>	Install application-specific LOAD DATA LOCAL INFILE handler callbacks.
<u>mysql_set_server_option()</u>	Sets an option for the connection (like multi-statements).

<u>mysql_sqlstate()</u>	Returns the SQLSTATE error code for the last error.
<u>mysql_shutdown()</u>	Shuts down the database server.
<u>mysql_stat()</u>	Returns the server status as a string.
<u>mysql_store_result()</u>	Retrieves a complete result set to the client.
<u>mysql_thread_id()</u>	Returns the current thread ID.
<u>mysql_thread_safe()</u>	Returns 1 if the clients are compiled as thread-safe.
<u>mysql_use_result()</u>	Initiates a row-by-row result set retrieval.
<u>mysql_warning_count()</u>	Returns the warning count for the previous SQL statement.

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL library by calling `mysql_library_init()`. The library can be either the `mysqlclient` C client library or the `mysqld` embedded server library, depending on whether the application was linked with the `-libmysqlclient` or `-libmysqld` flag.
2. Initialize a connection handler by calling `mysql_init()` and connect to the server by calling `mysql_real_connect()`.
3. Issue SQL statements and process their results. (The following discussion provides more information about how to do this.)
4. Close the connection to the MySQL server by calling `mysql_close()`.
5. End use of the MySQL library by calling `mysql_library_end()`.

The purpose of calling `mysql_library_init()` and `mysql_library_end()` is to provide proper initialization and finalization of the MySQL library. For applications that are linked with the client library, they provide improved memory management. If you don't call `mysql_library_end()`, a block of memory remains allocated. (This does not increase the amount of memory used by the application, but some memory leak detectors will complain about it.) For applications that are linked with the embedded server, these calls start and stop the server.

`mysql_library_init()` and `mysql_library_end()` are available as of MySQL 5.0.3. These actually are `#define` symbols that make them equivalent to `mysql_server_init()` and `mysql_server_end()`, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL library no matter whether the application uses the `mysqlclient` or `mysqld` library. For older versions of MySQL, you can call `mysql_server_init()` and `mysql_server_end()` instead.

In a non-multi-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, a race condition is possible if `mysql_library_init()` is invoked by `mysql_init()` in a multi-threaded environment: `mysql_library_init()` is not thread-safe, so it should be called prior to any other client library call.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the hostname, username, and password). Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. As of MySQL 5.0.13, you can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior. When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL statements to the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null-terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-`SELECT` query (for example, `INSERT`, `UPDATE`, `DELETE`), you can find out how many rows were changed (affected) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`-like in that they return rows. These include `SHOW`,

DESCRIBE, and EXPLAIN. They should be treated the same way as SELECT statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have previously been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about the size of the data in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Client programs should choose the approach that is most appropriate for their requirements. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that because the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set because it maintains only one row at a time (and because there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you don't have random access to rows within the result set (you can only access rows sequentially), and you don't know how many rows are in the result set until you have retrieved them all. Furthermore, you **must** retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking

for.

The API makes it possible for clients to respond appropriately to statements (retrieving rows only as necessary) without knowing whether the statement is a SELECT. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the statement was a SELECT and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether a result was actually to be expected. If `mysql_field_count()` returns zero, the statement returned no data (indicating that it was an INSERT, UPDATE, DELETE, and so forth), and was not expected to return rows. If `mysql_field_count()` is non-zero, the statement should have returned rows, but didn't. This indicates that the statement was a SELECT that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` allow you to obtain information about the fields that make up the result set (the number of fields, their names and types, and so forth). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, MySQL provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the most recently invoked function that can succeed or fail, allowing you to determine when an error occurred and what it was.

22.2.3. C API Function Descriptions

In the descriptions here, a parameter or return value of NULL means NULL in the sense of the C programming language, not a MySQL NULL value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-NULL value to indicate success or a NULL value to indicate an error, and functions returning an integer return zero to indicate success or non-zero to indicate an error. Note that

“non-zero” means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                /* correct */
    ... error ...

if (result < 0)            /* incorrect */
    ... error ...

if (result == -1)         /* incorrect */
    ... error ...
```

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

22.2.3.1. `mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Description

Returns the number of rows changed by the last UPDATE, deleted by the last DELETE or inserted by the last INSERT statement. May be called immediately after `mysql_query()` for UPDATE, DELETE, or INSERT statements. For SELECT statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an UPDATE statement, no rows matched the WHERE clause in the query or that no query has yet been executed. -1 indicates that the query returned an error or that, for a SELECT query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`. Because `mysql_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

Errors

None.

Example

```
mysql_query(&mysql, "UPDATE products SET cost=cost*1.25 WHERE group=1  
printf("%ld products updated", (long) mysql_affected_rows(&mysql));
```

If you specify the flag `CLIENT_FOUND_ROWS` when connecting to **mysqld**, `mysql_affected_rows()` returns the number of rows matched by the `WHERE` statement for `UPDATE` statements. Otherwise, it returns the number of rows actually changed.

Note that when you use a `REPLACE` command, `mysql_affected_rows()` returns 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

If you use `INSERT ... ON DUPLICATE KEY UPDATE` to insert a row, `mysql_affected_rows()` returns 1 if the row is inserted as a new row and 2 if an existing row is updated.

22.2.3.2. mysql_autocommit()

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

Description

Sets autocommit mode on if mode is 1, off if mode is 0.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

22.2.3.3. mysql_change_user()

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const  
char *password, const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_change_user()` fails if the connected user cannot be authenticated or doesn't have permission to use the database. In this case, the user and database are not changed

The `db` parameter may be set to `NULL` if you don't want to have a default database.

This command always performs a `ROLLBACK` of any active transactions, closes all temporary tables, unlocks all locked tables and resets the state as if one had done a new connect. This happens even if the user didn't change.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`.

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

- ER_UNKNOWN_COM_ERROR

The MySQL server doesn't implement this command (probably an old server).

- ER_ACCESS_DENIED_ERROR

The user or password was wrong.

- ER_BAD_DB_ERROR

The database didn't exist.

- ER_DBACCESS_DENIED_ERROR

The user did not have access rights to the database.

- ER_WRONG_DB_NAME

The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
    fprintf(stderr, "Failed to change user. Error: %s\n",
            mysql_error(&mysql));
}
```

22.2.3.4. mysql_character_set_name()

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Description

Returns the default character set for the current connection.

Return Values

The default character set

Errors

None.

22.2.3.5. `mysql_close()`

```
void mysql_close(MYSQL *mysql)
```

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_connect()`.

Return Values

None.

Errors

None.

22.2.3.6. `mysql_commit()`

```
my_bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

As of MySQL 5.0.3, the action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is 2, the server performs a release after terminating a transaction and closes the client connection. The client program should call `mysql_close()` to close the connection from the client side.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

22.2.3.7. `mysql_connect()`

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd)
```

Description

This function is deprecated. It is preferable to use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()` with the difference that the connection parameter may be `NULL`. In this case, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you can't retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

Return Values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

22.2.3.8. `mysql_create_db()`

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Description

Creates the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `CREATE DATABASE` statement instead.

Return Values

Zero if the database was created successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
    fprintf(stderr, "Failed to create new database. Error: %s\n",
            mysql_error(&mysql));
}
```

22.2.3.9. `mysql_data_seek()`

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a query result set. The `offset` value is a row number and should be in the range from 0 to `mysql_num_rows(result)-1`.

This function requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

None.

Errors

None.

22.2.3.10. `mysql_debug()`

```
void mysql_debug(const char *debug)
```

Description

Does a `DEBUG_PUSH` with the given string. `mysql_debug()` uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See [Section E.1, “Debugging a MySQL Server”](#), and [Section E.2, “Debugging a MySQL Client”](#).

Return Values

None.

Errors

None.

Example

The call shown here causes the client library to generate a trace file in `/tmp/client.trace` on the client machine:

```
mysql_debug("d:t:0,/tmp/client.trace");
```

22.2.3.11. `mysql_drop_db()`

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the db parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `DROP DATABASE` statement instead.

Return Values

Zero if the database was dropped successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

Example

```
if(mysql_drop_db(&mysql, "my_database"))
    fprintf(stderr, "Failed to drop the database: Error: %s\n",
            mysql_error(&mysql));
```

22.2.3.12. `mysql_dump_debug_info()`

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write some debug information to the log. For this to work, the connected user must have the SUPER privilege.

Return Values

Zero if the command was successful. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.13. `mysql_eof()`

```
my_bool mysql_eof(MYSQL_RES *result)
```

Description

This function is deprecated. `mysql_errno()` or `mysql_error()` may be used instead.

`mysql_eof()` determines whether the last row of a result set has been read.

If you acquire a result set from a successful call to `mysql_store_result()`, the client receives the entire set in one operation. In this case, a NULL return from `mysql_fetch_row()` always means the end of the result set has been reached and it is unnecessary to call `mysql_eof()`. When used with `mysql_store_result()`,

`mysql_eof()` always returns true.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL` return value from `mysql_fetch_row()` does not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a non-zero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard MySQL error functions `mysql_errno()` and `mysql_error()`. Because those error functions provide the same information, their use is preferred over `mysql_eof()`, which is deprecated. (In fact, they provide more information, because `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return Values

Zero if no error occurred. Non-zero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql, "SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result)) // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard MySQL error functions:

```

mysql_query(&mysql, "SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql)) // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}

```

22.2.3.14. `mysql_errno()`

```
unsigned int mysql_errno(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysql_error.h`. Errors also are listed at [Appendix B, Error Codes and Messages](#).

Note that some functions like `mysql_fetch_row()` don't set `mysql_errno()` if they succeed.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_errno()` if they succeed.

Return Values

An error code value for the last `mysql_xxx()` call, if it failed. zero means no error occurred.

Errors

None.

22.2.3.15. `mysql_error()`

```
const char *mysql_error(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_error()` returns a null-terminated string containing the error message for the most recently invoked API function that failed. If a function didn't fail, the return value of `mysql_error()` may be the previous error or an empty string to indicate no error.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_error()` if they succeed.

For functions that reset `mysql_errno()`, the following two tests are equivalent:

```
if(mysql_errno(&mysql))
{
    // an error occurred
}

if(mysql_error(&mysql)[0] != '\0')
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages. See [Section 5.11.2, “Setting the Error Message Language”](#).

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

22.2.3.16. `mysql_escape_string()`

You should use `mysql_real_escape_string()` instead!

This function is identical to `mysql_real_escape_string()` except that `mysql_real_escape_string()` takes a connection handler as its first argument and escapes the string according to the current character set.

`mysql_escape_string()` does not take a connection argument and does not respect the current character set.

22.2.3.17. `mysql_fetch_field()`

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, MySQL returns the default blob length (8KB) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8KB size is chosen because MySQL doesn't know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

Return Values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left.

Errors

None.

Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

22.2.3.18. `mysql_fetch_field_direct()`

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned  
int fieldnr)
```

Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. You may use this function to retrieve the definition for an arbitrary column. The value of `fieldnr` should be in the range from 0 to `mysql_num_fields(result)-1`.

Return Values

The `MYSQL_FIELD` structure for the specified column.

Errors

None.

Example

```
unsigned int num_fields;  
unsigned int i;  
MYSQL_FIELD *field;  
  
num_fields = mysql_num_fields(result);  
for(i = 0; i < num_fields; i++)  
{  
    field = mysql_fetch_field_direct(result, i);  
    printf("Field %u is %s\n", i, field->name);  
}
```

22.2.3.19. `mysql_fetch_fields()`

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

Return Values

An array of `MYSQL_FIELD` structures for all columns of a result set.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
    printf("Field %u is %s\n", i, fields[i].name);
}
```

22.2.3.20. `mysql_fetch_lengths()`

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you **must** use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

Return Values

An array of unsigned long integers representing the size of each column (not including any terminating null characters). `NULL` if an error occurred.

Errors

`mysql_fetch_lengths()` is valid only for the current row of the result set. It returns `NULL` if you call it before calling `mysql_fetch_row()` or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("Column %u is %lu bytes in length.\n", i, lengths[i])
    }
}
```

22.2.3.21. `mysql_fetch_row()`

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

Description

Retrieves the next row of a result set. When used after `mysql_store_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve. When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by `mysql_num_fields(result)`. If `row` holds the return value from a call to `mysql_fetch_row()`, pointers to the values are accessed as `row[0]` to `row[mysql_num_fields(result)-1]`. `NULL` values in the row are indicated by `NULL` pointers.

The lengths of the field values in the row may be obtained by calling `mysql_fetch_lengths()`. Empty fields and fields containing `NULL` both have length 0; you can distinguish these by checking the pointer for the field value. If

the pointer is NULL, the field is NULL; otherwise, the field is empty.

Return Values

A MYSQL_ROW structure for the next row. NULL if there are no more rows to retrieve or if an error occurred.

Errors

Note that error is not reset between calls to `mysql_fetch_row()`

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
    unsigned long *lengths;
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
        printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL")
    }
    printf("\n");
}
```

22.2.3.22. `mysql_field_count()`

```
unsigned int mysql_field_count(MYSQL *mysql)
```

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a non-empty result. This allows the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See [Section 22.2.13.1, “Why `mysql_store_result\(\)` Sometimes Returns `NULL` After `mysql_query\(\)` Returns Success”](#).

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

```

    }
    else // mysql_store_result() should have returned data
    {
        fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
    }
}
}

```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether the statement was a `SELECT`.

22.2.3.23. `mysql_field_seek()`

```

MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result,
MYSQL_FIELD_OFFSET offset)

```

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` retrieves the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an offset value of zero.

Return Values

The previous value of the field cursor.

Errors

None.

22.2.3.24. `mysql_field_tell()`

```

MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)

```

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

Return Values

The current offset of the field cursor.

Errors

None.

22.2.3.25. `mysql_free_result()`

```
void mysql_free_result(MYSQL_RES *result)
```

Description

Frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, and so forth. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

Do not attempt to access a result set after freeing it.

Return Values

None.

Errors

None.

22.2.3.26. `mysql_get_character_set_info()`

```
void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)
```

Description

This function provides information about the default client character set. The default character set may be changed with the `mysql_set_character_set()` function.

This function was added in MySQL 5.0.10.

Example

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set name: %s\n", cs.name);
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

22.2.3.27. `mysql_get_client_info()`

```
char *mysql_get_client_info(void)
```

Description

Returns a string that represents the client library version.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

22.2.3.28. `mysql_get_client_version()`

```
unsigned long mysql_get_client_version(void)
```

Description

Returns an integer that represents the client library version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level, and `ZZ` is the version number within the release level. For example, a value of `40102` represents a client library version of `4.1.2`.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

22.2.3.29. `mysql_get_host_info()`

```
char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server hostname.

Return Values

A character string representing the server hostname and the connection type.

Errors

None.

22.2.3.30. `mysql_get_proto_info()`

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Description

Returns the protocol version used by current connection.

Return Values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

22.2.3.31. `mysql_get_server_info()`

```
char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the server version number.

Return Values

A character string that represents the server version number.

Errors

None.

22.2.3.32. `mysql_get_server_version()`

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns the version number of the server as an integer.

Return Values

A number that represents the MySQL server version in this format:

```
major_version*10000 + minor_version *100 + sub_version
```

For example, 5.0.12 is returned as 50012.

This function is useful in client programs for quickly determining whether some version-specific server capability exists.

Errors

None.

22.2.3.33. `mysql_hex_string()`

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned
long length)
```

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See [Section 9.1.1, “Strings”](#).

The string in `from` is encoded to hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in `to` and a terminating null byte is appended.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_hex_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

The return value can be placed into an SQL statement using either `0xvalue` or `x'value'` format. However, the return value does not include the `0x` or `x' . . . '`. The caller must supply whichever of those is desired.

Example

```
char query[1000], *end;

end = strmov(query, "INSERT INTO test_table values(");
end = strmov(end, "0x");
end += mysql_hex_string(end, "What's this", 11);
end = strmov(end, ", 0x");
end += mysql_hex_string(end, "binary data: \0\r\n", 16);
*end++ = ')';

if (mysql_real_query(&mysql, query, (unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\n",
            mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

22.2.3.34. `mysql_info()`

```
char *mysql_info(MYSQL *mysql)
```

Description

Retrieves a string providing information about the most recently executed query, but only for the statements listed here. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of query, as described here. The numbers are illustrative only; the string contains values appropriate for the query.

- `INSERT INTO ... SELECT ...`

String format: Records: 100 Duplicates: 0 Warnings: 0

- `INSERT INTO ... VALUES (...),(...),(...)...`

String format: Records: 3 Duplicates: 0 Warnings: 0

- `LOAD DATA INFILE ...`

String format: Records: 1 Deleted: 0 Skipped: 0 Warnings: 0

- `ALTER TABLE`

String format: Records: 3 Duplicates: 0 Warnings: 0

- `UPDATE`

String format: Rows matched: 40 Changed: 40 Warnings: 0

Note that `mysql_info()` returns a non-NULL value for `INSERT ... VALUES` only for the multiple-row form of the statement (that is, only if multiple value lists are specified).

Return Values

A character string representing additional information about the most recently executed query. NULL if no information is available for the query.

Errors

None.

22.2.3.35. `mysql_init()`

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a MYSQL object suitable for `mysql_real_connect()`. If `mysql` is a NULL pointer, the function allocates, initializes, and returns a new object. Otherwise, the object is initialized and the address of the object is returned. If `mysql_init()` allocates a new object, it is freed when `mysql_close()` is called to close the connection.

Return Values

An initialized MYSQL* handle. NULL if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, NULL is returned.

22.2.3.36. `mysql_insert_id()`

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` or `UPDATE` statement. Use this function after you have performed an `INSERT` statement into a table that contains an `AUTO_INCREMENT` field.

More precisely, `mysql_insert_id()` is updated under these conditions:

- `INSERT` statements that store a value into an `AUTO_INCREMENT` column. This is true whether the value is automatically generated by storing the special values `NULL` or `0` into the column, or is an explicit non-special value.
- In the case of a multiple-row `INSERT` statement, `mysql_insert_id()` returns the **first** automatically generated `AUTO_INCREMENT` value; if no such value is generated, it returns the last **last** explicit value inserted into the `AUTO_INCREMENT` column.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by inserting `LAST_INSERT_ID(expr)` into any column.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by updating any column to `LAST_INSERT_ID(expr)`.
- The value of `mysql_insert_id()` is not affected by statements such as `SELECT` that return a result set.
- If the previous statement returned an error, the value of `mysql_insert_id()` is undefined.

Note that `mysql_insert_id()` returns `0` if the previous statement does not use an `AUTO_INCREMENT` value. If you need to save the value for later, be sure to call `mysql_insert_id()` immediately after the statement that generates the value.

The value of `mysql_insert_id()` is affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

See [Section 12.9.3, “Information Functions”](#).

Also note that the value of the SQL `LAST_INSERT_ID()` function always contains the most recently generated `AUTO_INCREMENT` value, and is not reset between statements because the value of that function is maintained in the server. Another

difference is that `LAST_INSERT_ID()` is not updated if you set an `AUTO_INCREMENT` column to a specific non-special value.

The reason for the difference between `LAST_INSERT_ID()` and `mysql_insert_id()` is that `LAST_INSERT_ID()` is made easy to use in scripts while `mysql_insert_id()` tries to provide a little more exact information of what happens to the `AUTO_INCREMENT` column.

Return Values

Described in the preceding discussion.

Errors

None.

22.2.3.37. `mysql_kill()`

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by `pid`.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.38. `mysql_library_end()`

```
void mysql_library_end(void)
```

Description

This is a synonym for the `mysql_server_end()` function. It was added in MySQL 5.0.3.

See [Section 22.2.2, “C API Function Overview”](#), for usage information.

22.2.3.39. `mysql_library_init()`

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

This is a synonym for the `mysql_server_init()` function. It was added in MySQL 5.0.3. See [Section 22.2.12.1, “mysql_server_init\(\)”](#).

See [Section 22.2.2, “C API Function Overview”](#) for usage information.

22.2.3.40. `mysql_list_dbs()`

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters ‘%’ or ‘_’, or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW databases [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A MYSQL_RES result set for success. NULL if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_OUT_OF_MEMORY

Out of memory.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.41. mysql_list_fields()

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

Description

Returns a result set consisting of field names in the given table that match the simple regular expression specified by the *wild* parameter. *wild* may contain the wildcard characters ‘%’ or ‘_’, or may be a NULL pointer to match all fields. Calling `mysql_list_fields()` is similar to executing the query `SHOW COLUMNS FROM tbl_name [LIKE wild]`.

Note that it's recommended that you use `SHOW COLUMNS FROM tbl_name` instead of `mysql_list_fields()`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

22.2.3.42. `mysql_list_processes()`

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by **mysqladmin processlist** or a `SHOW PROCESSLIST` query.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.43. `mysql_list_tables()`

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters ‘%’ or ‘_’, or may be a `NULL` pointer to match all tables. Calling `mysql_list_tables()` is similar to executing the query `SHOW tables [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.44. `mysql_more_results()`

```
my_bool mysql_more_results(MYSQL *mysql)
```

Description

Returns true if more results exist from the currently executed query, and the application must call `mysql_next_result()` to fetch the results.

Return Values

TRUE (1) if more results exist. FALSE (0) if no more results exist.

In most cases, you can call `mysql_next_result()` instead to test whether more results exist and initiate retrieval if so.

See [Section 22.2.9, “C API Handling of Multiple Statement Execution”](#), and [Section 22.2.3.45, “mysql_next_result\(\)”](#).

Errors

None.

22.2.3.45. `mysql_next_result()`

```
int mysql_next_result(MYSQL *mysql)
```

Description

If more query results exist, `mysql_next_result()` reads the next query results

and returns the status back to application.

You must call `mysql_free_result()` for the preceding query if it returned a result set.

After calling `mysql_next_result()` the state of the connection is as if you had called `mysql_real_query()` or `mysql_query()` for the next query. This means that you can call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If `mysql_next_result()` returns an error, no other statements are executed and there are no more results to fetch.

If your program executes stored procedures with the `CALL SQL` statement, you *must* set the `CLIENT_MULTI_RESULTS` flag, either explicitly, or implicitly by setting `CLIENT_MULTI_STATEMENTS` when you call `mysql_real_connect()`. This is because each `CALL` returns a result to indicate the call status, in addition to any results sets that might be returned by statements executed within the procedure. In addition, because `CALL` can return multiple results, you should process those results using a loop that calls `mysql_next_result()` to determine whether there are more results.

For an example that shows how to use `mysql_next_result()`, see [Section 22.2.9, “C API Handling of Multiple Statement Execution”](#).

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order. For example if you didn't call `mysql_use_result()` for a previous result set.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.46. `mysql_num_fields()`

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

To pass a `MYSQL*` argument instead, use `unsigned int mysql_field_count(MYSQL *mysql)`.

Description

Returns the number of columns in a result set.

Note that you can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if `mysql_store_result()` or `mysql_use_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a non-empty result. This allows the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See [Section 22.2.13.1, “Why `mysql_store_result\(\)` Sometimes Returns `NULL` After `mysql_query\(\)` Returns Success”](#).

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql, query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result) // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
}
```

An alternative (if you know that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check whether `mysql_field_count(&mysql)` is = 0. This happens only if something went wrong.

22.2.3.47. `mysql_num_rows()`

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` does not return the correct value until all the rows in the result set have been retrieved.

Return Values

The number of rows in the result set.

Errors

None.

22.2.3.48. `mysql_options()`

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)
```

Description

Can be used to set extra connect options and affect behavior for a connection. This function may be called multiple times to set several options.

`mysql_options()` should be called after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, then `arg` should point to the value of the integer.

Possible option values:

Option	Argument Type	Function
<code>MYSQL_INIT_COMMAND</code>	<code>char *</code>	Command to execute when c to the MySQL server. Will automatically be re-executed

		reconnecting.
MYSQL_OPT_COMPRESS	Not used	Use the compressed client/server protocol.
MYSQL_OPT_CONNECT_TIMEOUT	unsigned int *	Connect timeout in seconds.
MYSQL_OPT_GUESS_CONNECTION	Not used	For an application linked against libmysqld, this allows the library to guess whether to use the embedded server or a remote server. "G" means that if the hostname is not localhost, it uses a remote server. This behavior is the default. MYSQL_OPT_USE_EMBEDDED_CONNECTION and MYSQL_OPT_USE_REMOTE_CONNECTION can be used to override it. This is ignored for applications linked against libmysqlclient.
MYSQL_OPT_LOCAL_INFILE	optional pointer to uint	If no pointer is given or if pointer points to an unsigned int ! command LOAD LOCAL INFILE enabled.
MYSQL_OPT_NAMED_PIPE	Not used	Use named pipes to connect MySQL server on NT.
MYSQL_OPT_PROTOCOL	unsigned int *	Type of protocol to use. Show of the enum values of mysql_protocol_type defined in mysql.h.
MYSQL_OPT_READ_TIMEOUT	unsigned int *	Timeout for reads from server (currently only on Windows connections).
MYSQL_OPT_RECONNECT	my_bool *	Enable or disable automatic reconnection to the server if connection is found to have been lost. Reconnect has been off by default since MySQL 5.0.3; this option is on in 5.0.13 and provides a way

		reconnection behavior explic
MYSQL_OPT_SET_CLIENT_IP	char *	For an application linked ag linked against libmysqld (w libmysqld compiled with authentication support), this the user is considered to hav connected from the specifie address (specified as a string authentication purposes. Thi ignored for applications link libmysqlclient.
MYSQL_OPT_SSL_VERIFY_SERVER_CERT	my_bool *	Enable or disable verificatio server's Common Name valu certificate against the hostna when connecting to the serve connection is rejected if ther mismatch. This feature can b prevent man-in-the-middle a Verification is disabled by de Added in MySQL 5.0.23.
MYSQL_OPT_USE_EMBEDDED_CONNECTION	Not used	For an application linked ag libmysqld, this forces the us embedded server for the con This option is ignored for ap linked against libmysqlclie
MYSQL_OPT_USE_REMOTE_CONNECTION	Not used	For an application linked ag libmysqld, this forces the us remote server for the connec option is ignored for applica linked against libmysqlclie
MYSQL_OPT_USE_RESULT	Not used	This option is unused.
MYSQL_OPT_WRITE_TIMEOUT	unsigned int *	Timeout for writes to server currently only on Windows (c connections).
MYSQL_READ_DEFAULT_FILE	char *	Read options from the name file instead of from my.cnf.
		Read options from the name

MYSQL_READ_DEFAULT_GROUP	char *	from my.cnf or the file specified by the option MYSQL_READ_DEFAULT_FILE.
MYSQL_REPORT_DATA_TRUNCATION	my_bool *	Enable or disable reporting of data truncation errors for prepared statements via MYSQL_BIND. (Default: enabled) Added in MySQL 4.1.1.
MYSQL_SECURE_AUTH	my_bool *	Whether to connect to a server that does not support the password authentication used in MySQL 4.1.1 and later.
MYSQL_SET_CHARSET_DIR	char *	The pathname to the directory that contains character set definition files.
MYSQL_SET_CHARSET_NAME	char *	The name of the character set that is the default character set.
MYSQL_SHARED_MEMORY_BASE_NAME	char *	Named shared-memory object for communication to server. Should be the same as the option --shared-memory-base-name used for the mysqld you want to connect to.

Note that the client group is always read if you use MYSQL_READ_DEFAULT_FILE or MYSQL_READ_DEFAULT_GROUP.

The specified group in the option file may contain the following options:

Option	Description
connect-timeout	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.
compress	Use the compressed client/server protocol.
database	Connect to this database if no database was specified in the connect command.
debug	Debug options.
disable-local-infile	Disable use of LOAD DATA LOCAL.
host	Default hostname.
	Command to execute when connecting to MySQL.

init-command	server. Will automatically be re-executed when reconnecting.
interactive-timeout	Same as specifying CLIENT_INTERACTIVE to mysql_real_connect(). See Section 22.2.3.51, “mysql_real_connect()” .
local-infile[=(0 1)]	If no argument or argument != 0 then enable use of LOAD DATA LOCAL.
max_allowed_packet	Max size of packet client can read from server.
multi-results	Allow multiple result sets from multiple-statement executions or stored procedures.
multi-statements	Allow the client to send multiple statements in a single string (separated by ‘;’).
password	Default password.
pipe	Use named pipes to connect to a MySQL server on NT.
protocol={TCP SOCKET PIPE MEMORY}	The protocol to use when connecting to the server.
port	Default port number.
return-found-rows	Tell mysql_info() to return found rows instead of updated rows when using UPDATE.
shared-memory-base-name=name	Shared-memory name to use to connect to server (default is "MYSQL").
socket	Default socket file.
user	Default user.

Note that timeout has been replaced by connect-timeout, but timeout is still supported in MySQL 5.0.25 for backward compatibility.

For more information about option files, see [Section 4.3.2, “Using Option Files”](#).

Return Values

Zero for success. Non-zero if you used an unknown option.

Example

```

MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql, MYSQL_OPT_COMPRESS, 0);
mysql_options(&mysql, MYSQL_READ_DEFAULT_GROUP, "odbc");
if (!mysql_real_connect(&mysql, "host", "user", "passwd", "database", 0, N
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}

```

This code requests the client to use the compressed client/server protocol and read the additional options from the odbc section in the my.cnf file.

22.2.3.49. `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If the connection has gone down, an automatic reconnection is attempted.

This function can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

Return Values

Zero if the connection to the server is alive. Non-zero if an error occurred. A non-zero return does not indicate whether the MySQL server itself is down; the connection might be broken for other reasons such as network problems.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.50. `mysql_query()`

```
int mysql_query(MYSQL *mysql, const char *query)
```

Description

Executes the SQL query pointed to by the null-terminated string query. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (;) or \g to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See [Section 22.2.9, “C API Handling of Multiple Statement Execution”](#).

`mysql_query()` cannot be used for queries that contain binary data; you should use `mysql_real_query()` instead. (Binary data may contain the '\0' character, which `mysql_query()` interprets as the end of the query string.)

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See [Section 22.2.3.22, “mysql_field_count\(\)”](#).

Return Values

Zero if the query was successful. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.51. `mysql_real_connect()`

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd, const char *db, unsigned int port, const char *unix_socket, unsigned long client_flag)
```

Description

`mysql_real_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_real_connect()` must complete successfully before you can execute any other API functions that require a valid MySQL connection handle structure.

The parameters are specified as follows:

- The first parameter should be the address of an existing MySQL structure. Before calling `mysql_real_connect()` you must call `mysql_init()` to initialize the MySQL structure. You can change a lot of connect options with the `mysql_options()` call. See [Section 22.2.3.48, “mysql_options\(\)”](#).
- The value of `host` may be either a hostname or an IP address. If `host` is NULL or the string "localhost", a connection to the local host is assumed. If the OS supports sockets (Unix) or named pipes (Windows), they are used instead of TCP/IP to connect to the server.
- The `user` parameter contains the user's MySQL login ID. If `user` is NULL or the empty string "", the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current username must be specified explicitly. See the MyODBC section of [Chapter 23, Connectors](#).
- The `passwd` parameter contains the password for `user`. If `passwd` is NULL, only entries in the user table for the user that have a blank (empty) password field are checked for a match. This allows the database administrator to set up the MySQL privilege system in such a way that

users get different privileges depending on whether they have specified a password.

Note: Do not attempt to encrypt the password before calling `mysql_real_connect()`; password encryption is handled automatically by the client API.

- `db` is the database name. If `db` is not `NULL`, the connection sets the default database to this value.
- If `port` is not 0, the value is used as the port number for the TCP/IP connection. Note that the `host` parameter determines the type of the connection.
- If `unix_socket` is not `NULL`, the string specifies the socket or named pipe that should be used. Note that the `host` parameter determines the type of the connection.
- The value of `client_flag` is usually 0, but can be set to a combination of the following flags to enable certain features:

Flag Name	Flag Description
<code>CLIENT_COMPRESS</code>	Use compression protocol.
<code>CLIENT_FOUND_ROWS</code>	Return the number of found (matched) rows, not the number of changed rows.
<code>CLIENT_IGNORE_SPACE</code>	Allow spaces after function names. Makes all functions names reserved words.
<code>CLIENT_INTERACTIVE</code>	Allow <code>interactive_timeout</code> seconds (instead of <code>wait_timeout</code> seconds) of inactivity before closing the connection. The client's session <code>wait_timeout</code> variable is set to the value of the session <code>interactive_timeout</code> variable.
<code>CLIENT_LOCAL_FILES</code>	Enable <code>LOAD DATA LOCAL</code> handling.
<code>CLIENT_MULTI_STATEMENTS</code>	Tell the server that the client may send multiple statements in a single string (separated by <code>;</code>). If this flag is not set, multiple-statement execution is disabled. See

	the note following this table for more information about this flag.
CLIENT_MULTI_RESULTS	Tell the server that the client can handle multiple result sets from multiple-statement executions or stored procedures. This is automatically set if CLIENT_MULTI_STATEMENTS is set. See the note following this table for more information about this flag.
CLIENT_NO_SCHEMA	Don't allow the <i>db_name.tbl_name.col_name</i> syntax. This is for ODBC. It causes the parser to generate an error if you use that syntax, which is useful for trapping bugs in some ODBC programs.
CLIENT_ODBC	The client is an ODBC client. This changes mysqld to be more ODBC-friendly.
CLIENT_SSL	Use SSL (encrypted protocol). This option should not be set by application programs; it is set internally in the client library. Instead, use <code>mysql_ssl_set()</code> before calling <code>mysql_real_connect()</code> .

If your program executes stored procedures with the CALL SQL statement, you *must* set the CLIENT_MULTI_RESULTS flag, either explicitly, or implicitly by setting CLIENT_MULTI_STATEMENTS when you call `mysql_real_connect()`. This is because each CALL returns a result to indicate the call status, in addition to any results sets that might be returned by statements executed within the procedure.

If you enable CLIENT_MULTI_STATEMENTS or CLIENT_MULTI_RESULTS, you should process the result for every call to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 22.2.9, “C API Handling of Multiple Statement Execution”](#).

For some parameters, it is possible to have the value taken from an option file rather than from an explicit value in the `mysql_real_connect()` call. To do this, call `mysql_options()` with the MYSQL_READ_DEFAULT_FILE or MYSQL_READ_DEFAULT_GROUP option before calling `mysql_real_connect()`.

Then, in the `mysql_real_connect()` call, specify the “no-value” value for each parameter to be read from an option file:

- For `host`, specify a value of `NULL` or the empty string (`""`).
- For `user`, specify a value of `NULL` or the empty string.
- For `passwd`, specify a value of `NULL`. (For the password, a value of the empty string in the `mysql_real_connect()` call cannot be overridden in an option file, because the empty string indicates explicitly that the MySQL account must have an empty password.)
- For `db`, specify a value of `NULL` or the empty string.
- For `port`, specify a value of `0`.
- For `unix_socket`, specify a value of `NULL`.

If no value is found in an option file for a parameter, its default value is used as indicated in the descriptions given earlier in this section.

Return Values

A `MYSQL*` connection handle if the connection was successful, `NULL` if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter.

Errors

- `CR_CONN_HOST_ERROR`

Failed to connect to the MySQL server.

- `CR_CONNECTION_ERROR`

Failed to connect to the local MySQL server.

- `CR_IPSOCK_ERROR`

Failed to create an IP socket.

- CR_OUT_OF_MEMORY

Out of memory.

- CR_SOCKET_CREATE_ERROR

Failed to create a Unix socket.

- CR_UNKNOWN_HOST

Failed to find the IP address for the hostname.

- CR_VERSION_ERROR

A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version. This can happen if you use a very old client library to connect to a new server that wasn't started with the `--old-protocol` option.

- CR_NAMEDPIPEOPEN_ERROR

Failed to create a named pipe on Windows.

- CR_NAMEDPIPEWAIT_ERROR

Failed to wait for a named pipe on Windows.

- CR_NAMEDPIPESETSTATE_ERROR

Failed to get a pipe handler on Windows.

- CR_SERVER_LOST

If `connect_timeout > 0` and it took longer than `connect_timeout` seconds to connect to the server or if the server died while executing the `init-command`.

Example

```
MYSQL mysql;
```

```
mysql_init(&mysql);
```

```
mysql_options(&mysql, MYSQL_READ_DEFAULT_GROUP, "your_prog_name");
if (!mysql_real_connect(&mysql, "host", "user", "passwd", "database", 0, N
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}
```

By using `mysql_options()` the MySQL library reads the `[client]` and `[your_prog_name]` sections in the `my.cnf` file which ensures that your program works, even if someone has set up MySQL in some non-standard way.

Note that upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than `5.0.3`, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. As of MySQL `5.0.13`, you can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior.

22.2.3.52. `mysql_real_escape_string()`

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to,
const char *from, unsigned long length)
```

Note that `mysql` must be a valid, open connection. This is needed because the escaping depends on the character set in use by the server.

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See [Section 9.1.1, “Strings”](#).

The string in `from` is encoded to an escaped SQL string, taking into account the current character set of the connection. The result is placed in `to` and a terminating null byte is appended. Characters encoded are `NUL` (ASCII `0`), `'\n'`, `'\r'`, `'\'`, `'\''`, `'\"'`, and Control-Z (see [Section 9.1, “Literal Values”](#)). (Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. This function quotes the other characters to make them easier to read in log files.)

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. (In the worst case, each character

may need to be encoded as using two bytes, and you need room for the terminating null byte.) When `mysql_real_escape_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

If you need to change the character set of the connection, you should use the `mysql_set_character_set()` function rather than executing a `SET NAMES` (or `SET CHARACTER SET`) statement. `mysql_set_character_set()` works like `SET NAMES` but also affects the character set used by `mysql_real_escape_string()`, which `SET NAMES` does not.

Example

```
char query[1000], *end;

end = strmov(query, "INSERT INTO test_table values(");
*end++ = '\\';
end += mysql_real_escape_string(&mysql, end, "what's this", 11);
*end++ = '\\';
*end++ = ',';
*end++ = '\\';
end += mysql_real_escape_string(&mysql, end, "binary data: \\0\\r\\n", 16);
*end++ = '\\';
*end++ = ')';

if (mysql_real_query(&mysql, query, (unsigned int) (end - query)))
{
    fprintf(stderr, "Failed to insert row, Error: %s\\n",
            mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

22.2.3.53. `mysql_real_query()`

```
int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)
```

Description

Executes the SQL query pointed to by `query`, which should be a string `length` bytes long. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (‘;’) or `\g` to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See [Section 22.2.9, “C API Handling of Multiple Statement Execution”](#).

You **must** use `mysql_real_query()` rather than `mysql_query()` for queries that contain binary data, because binary data may contain the ‘\0’ character. In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the query string.

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See [Section 22.2.3.22, “`mysql_field_count\(\)`”](#).

Return Values

Zero if the query was successful. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.54. `mysql_refresh()`

```
int mysql_refresh(MYSQL *mysql, unsigned int options)
```

Description

This function flushes tables or caches, or resets replication server information. The connected user must have the RELOAD privilege.

The options argument is a bit mask composed from any combination of the following values. Multiple values can be OR'ed together to perform multiple operations with a single call.

- REFRESH_GRANT

Refresh the grant tables, like `FLUSH PRIVILEGES`.

- REFRESH_LOG

Flush the logs, like `FLUSH LOGS`.

- REFRESH_TABLES

Flush the table cache, like `FLUSH TABLES`.

- REFRESH_HOSTS

Flush the host cache, like `FLUSH HOSTS`.

- REFRESH_STATUS

Reset status variables, like `FLUSH STATUS`.

- REFRESH_THREADS

Flush the thread cache.

- REFRESH_SLAVE

On a slave replication server, reset the master server information and restart the slave, like RESET SLAVE.

- REFRESH_MASTER

On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like RESET MASTER.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.55. mysql_reload()

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the MySQL server to reload the grant tables. The connected user must have the RELOAD privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `FLUSH PRIVILEGES` statement instead.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

22.2.3.56. `mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

As of MySQL 5.0.3, the action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is 2, the server performs a release after terminating a transaction and closes the client connection. The client program should call `mysql_close()` to close the connection from the client side.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

22.2.3.57. `mysql_row_seek()`

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET  
offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. The offset value is a row offset that should be a value returned from `mysql_row_tell()` or from `mysql_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

22.2.3.58. `mysql_row_tell()`

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

You should use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

22.2.3.59. `mysql_select_db()`

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.3.60. `mysql_set_character_set()`

```
int mysql_set_character_set(MYSQL *mysql, char *csname)
```

Description

This function is used to set the default character set for the current connection. The string `csname` specifies a valid character set name. The connection collation becomes the default collation of the character set. This function works like the `SET NAMES` statement, but also sets the value of `mysql->charset`, and thus affects the character set used by `mysql_real_escape_string()`

This function was added in MySQL 5.0.7.

Return Values

Zero for success. Non-zero if an error occurred.

Example

```
MYSQL mysql;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql, "host", "user", "passwd", "database", 0, N
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
            mysql_error(&mysql));
}

if (!mysql_set_character_set(&mysql, "utf8"))
{
    printf("New client character set: %s\n", mysql_character_set_name
}
}
```

22.2.3.61. `mysql_set_local_infile_default()`

```
void
mysql_set_local_infile_default(MYSQL *mysql);
```

Description

Sets the `LOAD LOCAL DATA INFILE` handler callback functions to the defaults used internally by the C client library. The library calls this function automatically if `mysql_set_local_infile_handler()` has not been called or does not supply valid functions for each of its callbacks.

The `mysql_set_local_infile_default()` function was added in MySQL 4.1.2.

Return Values

None.

Errors

None.

22.2.3.62. `mysql_set_local_infile_handler()`

```
void
mysql_set_local_infile_handler(MYSQL *mysql,
                               int (*local_infile_init)(void **, const char *, void
                                                       int (*local_infile_read)(void *, char *, unsigned int
                                                       void (*local_infile_end)(void *),
                                                       int (*local_infile_error)(void *, char*, unsigned int
                                                       void *userdata);
```

Description

This function installs callbacks to be used during the execution of `LOAD DATA LOCAL INFILE` statements. It enables application programs to exert control over local (client-side) datafile reading. The arguments are the connection handler, a set of pointers to callback functions, and a pointer to a data area that the callbacks can use to share information.

To use `mysql_set_local_infile_handler()`, you must write the following callback functions:

```
int
local_infile_init(void **ptr, const char *filename, void *userdata);
```

The initialization function. This is called once to do any setup necessary, open

the datafile, allocate data structures, and so forth. The first `void**` argument is a pointer to a pointer. You can set the pointer (that is, `*ptr`) to a value that will be passed to each of the other callbacks (as a `void*`). The callbacks can use this pointed-to value to maintain state information. The `userdata` argument is the same value that is passed to `mysql_set_local_infile_handler()`.

The initialization function should return zero for success, non-zero for an error.

```
int
local_infile_read(void *ptr, char *buf, unsigned int buf_len);
```

The data-reading function. This is called repeatedly to read the data file. `buf` points to the buffer where the read data should be stored, and `buf_len` is the maximum number of bytes that the callback can read and store in the buffer. (It can read fewer bytes, but should not read more.)

The return value is the number of bytes read, or zero when no more data could be read (this indicates EOF). Return a value less than zero if an error occurs.

```
void
local_infile_end(void *ptr)
```

The termination function. This is called once after `local_infile_read()` has returned zero (EOF) or an error. This function should deallocate any memory allocated by `local_infile_init()` and perform any other cleanup necessary. It is invoked even if the initialization function returns an error.

```
int
local_infile_error(void *ptr, char *error_msg, unsigned int error_ms
```

The error-handling function. This is called to get a textual error message to return to the user in case any of your other functions returns an error. `error_msg` points to the buffer into which the message should be written, and `error_msg_len` is the length of the buffer. The message should be written as a null-terminated string, so the message can be at most `error_msg_len-1` bytes long.

The return value is the error number.

Typically, the other callbacks store the error message in the data structure pointed to by `ptr`, so that `local_infile_error()` can copy the message from

there into `error_msg`.

After calling `mysql_set_local_infile_handler()` in your C code and passing pointers to your callback functions, you can then issue a `LOAD DATA LOCAL INFILE` statement (for example, by using `mysql_query()`). The client library automatically invokes your callbacks. The filename specified in `LOAD DATA LOCAL INFILE` will be passed as the second parameter to the `local_infile_init()` callback.

The `mysql_set_local_infile_handler()` function was added in MySQL 4.1.2.

Return Values

None.

Errors

None.

22.2.3.63. `mysql_set_server_option()`

```
int mysql_set_server_option(MYSQL *mysql, enum
enum_mysql_set_option option)
```

Description

Enables or disables an option for the connection. `option` can have one of the following values:

<code>MYSQL_OPTION_MULTI_STATEMENTS_ON</code>	Enable multiple-statement support.
<code>MYSQL_OPTION_MULTI_STATEMENTS_OFF</code>	Disable multiple-statement support.

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see [Section 22.2.9, “C API Handling of Multiple Statement Execution”](#).

Enabling multiple-statement support with `MYSQL_OPTION_MULTI_STATEMENTS_ON` does not have quite the same effect as enabling it by passing the `CLIENT_MULTI_STATEMENTS` flag to `mysql_real_connect()`: `CLIENT_MULTI_STATEMENTS` also enables `CLIENT_MULTI_RESULTS`. If you are using the `CALL SQL` statement in your programs, multiple-result support must be enabled; this means that `MYSQL_OPTION_MULTI_STATEMENTS_ON` by itself is insufficient to allow the use of `CALL`.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `ER_UNKNOWN_COM_ERROR`

The server didn't support `mysql_set_server_option()` (which is the case that the server is older than 4.1.1) or the server didn't support the option one tried to set.

22.2.3.64. `mysql_shutdown()`

```
int mysql_shutdown(MYSQL *mysql, enum enum_shutdown_level shutdown_level)
```

Description

Asks the database server to shut down. The connected user must have `SHUTDOWN`

privileges. The `shutdown_level` argument was added in MySQL 5.0.1. MySQL 5.0 servers support only one type of shutdown; `shutdown_level` must be equal to `SHUTDOWN_DEFAULT`. Additional shutdown levels are planned to make it possible to choose the desired level. Dynamically linked executables which have been compiled with older versions of the `libmysqlclient` headers and call `mysql_shutdown()` need to be used with the old `libmysqlclient` dynamic library.

The shutdown process is described in [Section 5.2.6, “The MySQL Server Shutdown Process”](#).

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

22.2.3.65. `mysql_sqlstate()`

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the `SQLSTATE` error code for the

last error. The error code consists of five characters. '00000' means “no error.” The values are specified by ANSI SQL and ODBC. For a list of possible values, see [Appendix B, Error Codes and Messages](#).

Note that not all MySQL errors are mapped to SQLSTATE error codes. The value 'HY000' (general error) is used for unmapped errors.

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See [Section 22.2.3.14, “mysql_errno\(\)”](#), [Section 22.2.3.15, “mysql_error\(\)”](#), and [Section 22.2.7.26, “mysql_stmt_sqlstate\(\)”](#).

22.2.3.66. mysql_ssl_set()

```
int mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert,
const char *ca, const char *capath, const char *cipher)
```

Description

`mysql_ssl_set()` is used for establishing secure connections using SSL. It must be called before `mysql_real_connect()`.

`mysql_ssl_set()` does nothing unless OpenSSL support is enabled in the client library.

`mysql` is the connection handler returned from `mysql_init()`. The other parameters are specified as follows:

- `key` is the pathname to the key file.
- `cert` is the pathname to the certificate file.
- `ca` is the pathname to the certificate authority file.
- `capath` is the pathname to a directory that contains trusted SSL CA certificates in pem format.

- `cipher` is a list of allowable ciphers to use for SSL encryption.

Any unused SSL parameters may be given as `NULL`.

Return Values

This function always returns `0`. If SSL setup is incorrect, `mysql_real_connect()` returns an error when you attempt to connect.

22.2.3.67. `mysql_stat()`

```
char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the **mysqladmin status** command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Return Values

A character string describing the server status. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

22.2.3.68. `mysql_store_result()`

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query that successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth).

You don't have to call `mysql_store_result()` or `mysql_use_result()` for other queries, but it does not do any harm or cause any notable performance degradation if you call `mysql_store_result()` in all cases. You can detect if the query didn't have a result set by checking if `mysql_store_result()` returns 0 (more about this later on).

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See [Section 22.2.3.22, “`mysql_field_count\(\)`”](#).

`mysql_store_result()` reads the entire result of a query to the client, allocates a `MYSQL_RES` structure, and places the result into this structure.

`mysql_store_result()` returns a null pointer if the query didn't return a result set (if the query was, for example, an `INSERT` statement).

`mysql_store_result()` also returns a null pointer if reading of the result set failed. You can check whether an error occurred by checking if `mysql_error()` returns a non-empty string, if `mysql_errno()` returns non-zero, or if `mysql_field_count()` returns zero.

An empty result set is returned if there are no rows returned. (An empty result set differs from a null pointer as a return value.)

Once you have called `mysql_store_result()` and got a result back that isn't a null pointer, you may call `mysql_num_rows()` to find out how many rows are in the result set.

You can call `mysql_fetch_row()` to fetch rows from the result set, or `mysql_row_seek()` and `mysql_row_tell()` to obtain or set the current row

position within the result set.

You must call `mysql_free_result()` once you are done with the result set.

See [Section 22.2.13.1, “Why `mysql_store_result\(\)` Sometimes Returns `NULL` After `mysql_query\(\)` Returns Success”](#).

Return Values

A `MYSQL_RES` result structure with the results. `NULL` if an error occurred.

Errors

`mysql_store_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

22.2.3.69. `mysql_thread_id()`

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID changes. This means you should not get the thread ID and store it for later. You should get it when you need it.

Return Values

The thread ID of the current connection.

Errors

None.

22.2.3.70. `mysql_use_result()`

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query that successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client allocates memory only for the current row and a communication buffer that may grow up to `max_allowed_packet` bytes.

On the other hand, you shouldn't use `mysql_use_result()` if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This ties up the server and prevent other threads from updating any tables from which the data is being fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise, the unfetched rows are returned as part of the

result set for your next query. The C API gives the error Commands out of sync; you can't run this command now if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()`, or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other queries until `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` accurately returns the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

When using the `libmysqld` embedded server, the memory benefits are essentially lost because memory usage incrementally increases with each row retrieved until `mysql_free_result()` is called.

Return Values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

Errors

`mysql_use_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

22.2.3.71. `mysql_warning_count()`

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of warnings generated during execution of the previous SQL statement.

Return Values

The warning count.

Errors

None.

22.2.4. C API Prepared Statements

The MySQL client/server protocol provides for the use of prepared statements. This capability uses the `MYSQL_STMT` statement handler data structure returned by the `mysql_stmt_init()` initialization function. Prepared execution is an efficient way to execute a statement more than once. The statement is first parsed to prepare it for execution. Then it is executed one or more times at a later time, using the statement handle returned by the initialization function.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Prepared statements might not provide a performance increase in some situations. For best results, test your application both with prepared and non-prepared statements and choose whichever yields best performance.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

The following statements can be used as prepared statements: CREATE TABLE, DELETE, DO, INSERT, REPLACE, SELECT, SET, UPDATE, and most SHOW statements. Other statements are not supported in MySQL 5.0.

22.2.5. C API Prepared Statement Data types

Prepared statements mainly use the MYSQL_STMT and MYSQL_BIND data structures. A third structure, MYSQL_TIME, is used to transfer temporal data.

- **MYSQL_STMT**

This structure represents a prepared statement. A statement is created by calling `mysql_stmt_init()`, which returns a statement handle (that is, a pointer to a `MYSQL_STMT`). The handle is used for all subsequent statement-related functions until you close it with `mysql_stmt_close()`.

The `MYSQL_STMT` structure has no members that are for application use. Also, you should not try to make a copy of a `MYSQL_STMT` structure. There is no guarantee that such a copy will be usable.

Multiple statement handles can be associated with a single connection. The limit on the number of handles depends on the available system resources.

- **MYSQL_BIND**

This structure is used both for statement input (data values sent to the server) and output (result values returned from the server). For input, it is used with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`. For output, it is used with `mysql_stmt_bind_result()` to bind result set buffers for use in fetching rows with `mysql_stmt_fetch()`.

To use a `MYSQL_BIND` structure, you should zero its contents to initialize it, and then set its members appropriately. For example, to declare and initialize an array of three `MYSQL_BIND` structures, use this code:

```
MYSQL_BIND    bind[3];
memset(bind, 0, sizeof(bind));
```

The `MYSQL_BIND` structure contains the following members for use by

application programs. Each is used both for input and for output, although sometimes for different purposes depending on the direction of data transfer.

- `enum enum_field_types buffer_type`

The type of the buffer. The allowable `buffer_type` values are listed later in this section. For input, `buffer_type` indicates what type of value you are binding to a statement parameter. For output, it indicates what type of value you expect to receive in a result buffer.

- `void *buffer`

For input, this is a pointer to the buffer in which a statement parameter's data value is stored. For output, it is a pointer to the buffer in which to return a result set column value. For numeric data types, `buffer` should point to a variable of the proper C type. (If you are associating the variable with a column that has the `UNSIGNED` attribute, the variable should be an unsigned C type. Indicate whether the variable is signed or unsigned by using the `is_unsigned` member, described later in this list.) For date and time data types, `buffer` should point to a `MYSQL_TIME` structure. For character and binary string data types, `buffer` should point to a character buffer.

- `unsigned long buffer_length`

The actual size of `*buffer` in bytes. This indicates the maximum amount of data that can be stored in the buffer. For character and binary C data, the `buffer_length` value specifies the length of `*buffer` when used with `mysql_stmt_bind_param()`, or the maximum number of data bytes that can be fetched into the buffer when used with `mysql_stmt_bind_result()`.

- `unsigned long *length`

A pointer to an unsigned `long` variable that indicates the actual number of bytes of data stored in `*buffer`. `length` is used for character or binary C data. For input parameter data binding, `length` points to an unsigned `long` variable that indicates the length of the parameter value stored in `*buffer`; this is used by

`mysql_stmt_execute()`. For output value binding, the return value of `mysql_stmt_fetch()` determines the interpretation of the length. If `mysql_stmt_fetch()` returns 0, `*length` indicates the actual length of the parameter value. If `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED`, `*length` indicates the non-truncated length of the parameter value. In this case, the minimum of `*length` and `buffer_length` indicates the actual length of the value.

`length` is ignored for numeric and temporal data types because the length of the data value is determined by the `buffer_type` value.

- `my_bool *is_null`

This member points to a `my_bool` variable that is true if a value is `NULL`, false if it is not `NULL`. For input, set `*is_null` to true to indicate that you are passing a `NULL` value as a statement parameter. For output, this value is set to true after you fetch a row if the result set column value returned from the statement is `NULL`.

`is_null` is a pointer to a boolean rather than a boolean scalar so that it can be used in the following way:

- If your data values are always `NULL`, use `MYSQL_TYPE_NULL` to bind the column.
- If your data values are always `NOT NULL`, set `is_null = (my_bool*) 0`.
- In all other cases, you should set `is_null` to the address of a `my_bool` variable and change that variable's value appropriately between executions to indicate whether data values are `NULL` or `NOT NULL`.

- `my_bool is_unsigned`

This member is used for integer types. (These correspond to the `MYSQL_TYPE_TINY`, `MYSQL_TYPE_SHORT`, `MYSQL_TYPE_LONG`, and `MYSQL_TYPE_LONGLONG` type codes.) `is_unsigned` should be set to true for unsigned types and false for signed types.

- `my_bool *error`

For output, set this member to point to a `my_bool` variable to have truncation information for the parameter stored there after a row fetching operation. (Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.) When truncation reporting is enabled, `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED` and `*error` is true in the `MYSQL_BIND` structures for parameters in which truncation occurred. Truncation indicates loss of sign or significant digits, or that a string was too long to fit in a column. The `error` member was added in MySQL 5.0.3.

- `MYSQL_TIME`

This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. This is done by setting the `buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types, and setting the `buffer` member to point to a `MYSQL_TIME` structure.

The `MYSQL_TIME` structure contains the following members:

- `unsigned int year`

The year.

- `unsigned int month`

The month of the year.

- `unsigned int day`

The day of the month.

- `unsigned int hour`

The hour of the day.

- `unsigned int minute`

The minute of the hour.

- unsigned int second

The second of the minute.

- my_bool neg

A boolean flag to indicate whether the time is negative.

- unsigned long second_part

The fractional part of the second. This member currently is unused.

Only those parts of a `MYSQL_TIME` structure that apply to a given type of temporal value are used: The year, month, and day elements are used for `DATE`, `DATETIME`, and `TIMESTAMP` values. The hour, minute, and second elements are used for `TIME`, `DATETIME`, and `TIMESTAMP` values. See [Section 22.2.10, “C API Handling of Date and Time Values”](#).

The following table shows the allowable values that may be specified in the `buffer_type` member of `MYSQL_BIND` structures. The table also shows those SQL types that correspond most closely to each `buffer_type` value, and, for numeric and temporal types, the corresponding recommended C type.

The types are “recommended” because implicit type conversion may be performed in both directions. The `buffer_type` value controls the conversion that will be performed. For example, to fetch a SQL `MEDIUMINT` column value, you can specify a `buffer_type` value of `MYSQL_TYPE_LONG` and use a C variable of type `int` as the destination buffer. If you fetch a numeric column with a value of 255 into a `char[4]` character array, specify a `buffer_type` value of `MYSQL_TYPE_STRING` and the resulting value in the array will be a 4-byte string containing `'255\0'`.

To distinguish between binary and non-binary data for string data types, check whether the `charsetnr` value of the result set metadata is 63. If so, the character set is binary, which indicates binary rather than non-binary data. This is how to distinguish between `BINARY` and `CHAR`, `VARBINARY` and `VARCHAR`, and `BLOB` and `TEXT`.



buffer_type Value	SQL Type	Recommended C Type
MYSQL_TYPE_BIT	BIT	unsigned long long int
MYSQL_TYPE_TINY	TINYINT	unsigned char
MYSQL_TYPE_SHORT	SMALLINT	short int
MYSQL_TYPE_LONG	INT	int
MYSQL_TYPE_LONGLONG	BIGINT	long long int
MYSQL_TYPE_FLOAT	FLOAT	float
MYSQL_TYPE_DOUBLE	DOUBLE	double
MYSQL_TYPE_NEWDECIMAL	DECIMAL	char[]
MYSQL_TYPE_TIME	TIME	MYSQL_TIME
MYSQL_TYPE_DATE	DATE	MYSQL_TIME
MYSQL_TYPE_DATETIME	DATETIME	MYSQL_TIME
MYSQL_TYPE_TIMESTAMP	TIMESTAMP	MYSQL_TIME
MYSQL_TYPE_STRING	CHAR/BINARY	char[]
MYSQL_TYPE_VAR_STRING	VARCHAR/VARBINARY	char[]
MYSQL_TYPE_TINY_BLOB	TINYBLOB/TINYTEXT	char[]
MYSQL_TYPE_BLOB	BLOB/TEXT	char[]
MYSQL_TYPE_MEDIUM_BLOB	MEDIUMBLOB/MEDIUMTEXT	char[]
MYSQL_TYPE_LONG_BLOB	LOB/LOB	char[]

22.2.6. C API Prepared Statement Function Overview

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See [Section 22.2.7, “C API Prepared Statement Function Descriptions”](#).

Function	Description
mysql_stmt_affected_rows()	Returns the number of rows changes, deleted, or inserted by prepared UPDATE, DELETE, or INSERT statement.
mysql_stmt_attr_get()	Get value of an attribute for a prepared statement.
mysql_stmt_attr_set()	Sets an attribute for a prepared statement.
mysql_stmt_bind_param()	Associates application data buffers with the parameter markers in a prepared SQL statement.

<u>mysql_stmt_bind_result()</u>	Associates application data buffers with columns in the result set.
<u>mysql_stmt_close()</u>	Frees memory used by prepared statement.
<u>mysql_stmt_data_seek()</u>	Seeks to an arbitrary row number in a statement result set.
<u>mysql_stmt_errno()</u>	Returns the error number for the last statement execution.
<u>mysql_stmt_error()</u>	Returns the error message for the last statement execution.
<u>mysql_stmt_execute()</u>	Executes the prepared statement.
<u>mysql_stmt_fetch()</u>	Fetches the next row of data from the result set and returns data for all bound columns.
<u>mysql_stmt_fetch_column()</u>	Fetch data for one column of the current row of the result set.
<u>mysql_stmt_field_count()</u>	Returns the number of result columns for the most recent statement.
<u>mysql_stmt_free_result()</u>	Free the resources allocated to the statement handle.
<u>mysql_stmt_init()</u>	Allocates memory for MYSQL_STMT structure and initializes it.
<u>mysql_stmt_insert_id()</u>	Returns the ID generated for an AUTO_INCREMENT column by prepared statement.
<u>mysql_stmt_num_rows()</u>	Returns total rows from the statement buffered result set.
<u>mysql_stmt_param_count()</u>	Returns the number of parameters in a prepared SQL statement.
<u>mysql_stmt_param_metadata()</u>	(Return parameter metadata in the form of a result set.) Currently, this function does nothing.
<u>mysql_stmt_prepare()</u>	Prepares an SQL string for execution.
<u>mysql_stmt_reset()</u>	Reset the statement buffers in the server.
<u>mysql_stmt_result_metadata()</u>	Returns prepared statement metadata in the form of a result set.

<u>mysql_stmt_row_seek()</u>	Seeks to a row offset in a statement result set, using value returned from <code>mysql_stmt_row_tell()</code> .
<u>mysql_stmt_row_tell()</u>	Returns the statement row cursor position.
<u>mysql_stmt_send_long_data()</u>	Sends long data in chunks to server.
<u>mysql_stmt_sqlstate()</u>	Returns the SQLSTATE error code for the last statement execution.
<u>mysql_stmt_store_result()</u>	Retrieves the complete result set to the client.

Call `mysql_stmt_init()` to create a statement handle, then `mysql_stmt_prepare` to prepare it, `mysql_stmt_bind_param()` to supply the parameter data, and `mysql_stmt_execute()` to execute the statement. You can repeat the `mysql_stmt_execute()` by changing parameter values in the respective buffers supplied through `mysql_stmt_bind_param()`.

If the statement is a `SELECT` or any other statement that produces a result set, `mysql_stmt_prepare()` also returns the result set metadata information in the form of a `MYSQL_RES` result set through `mysql_stmt_result_metadata()`.

You can supply the result buffers using `mysql_stmt_bind_result()`, so that the `mysql_stmt_fetch()` automatically returns data to these buffers. This is row-by-row fetching.

You can also send the text or binary data in chunks to server using `mysql_stmt_send_long_data()`. See [Section 22.2.7.25, “`mysql_stmt_send_long_data\(\)`”](#).

When statement execution has been completed, the statement handle must be closed using `mysql_stmt_close()` so that all resources associated with it can be freed.

If you obtained a `SELECT` statement's result set metadata by calling `mysql_stmt_result_metadata()`, you should also free the metadata using `mysql_free_result()`.

Execution Steps

To prepare and execute a statement, an application follows these steps:

1. Create a prepared statement handle with `mysql_stmt_init()`. To prepare the statement on the server, call `mysql_stmt_prepare()` and pass it a string containing the SQL statement.
2. If the statement produces a result set, call `mysql_stmt_result_metadata()` to obtain the result set metadata. This metadata is itself in the form of result set, albeit a separate one from the one that contains the rows returned by the query. The metadata result set indicates how many columns are in the result and contains information about each column.
3. Set the values of any parameters using `mysql_stmt_bind_param()`. All parameters must be set. Otherwise, statement execution returns an error or produces unexpected results.
4. Call `mysql_stmt_execute()` to execute the statement.
5. If the statement produces a result set, bind the data buffers to use for retrieving the row values by calling `mysql_stmt_bind_result()`.
6. Fetch the data into the buffers row by row by calling `mysql_stmt_fetch()` repeatedly until no more rows are found.
7. Repeat steps 3 through 6 as necessary, by changing the parameter values and re-executing the statement.

When `mysql_stmt_prepare()` is called, the MySQL client/server protocol performs these actions:

- The server parses the statement and sends the okay status back to the client by assigning a statement ID. It also sends total number of parameters, a column count, and its metadata if it is a result set oriented statement. All syntax and semantics of the statement are checked by the server during this call.
- The client uses this statement ID for the further operations, so that the server can identify the statement from among its pool of statements.

When `mysql_stmt_execute()` is called, the MySQL client/server protocol performs these actions:

- The client uses the statement handle and sends the parameter data to the server.
- The server identifies the statement using the ID provided by the client, replaces the parameter markers with the newly supplied data, and executes the statement. If the statement produces a result set, the server sends the data back to the client. Otherwise, it sends an okay status and total number of rows changed, deleted, or inserted.

When `mysql_stmt_fetch()` is called, the MySQL client/server protocol performs these actions:

- The client reads the data from the packet row by row and places it into the application data buffers by doing the necessary conversions. If the application buffer type is same as that of the field type returned from the server, the conversions are straightforward.

If an error occurs, you can get the statement error code, error message, and `SQLSTATE` value using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

Prepared Statement Logging

For prepared statements that are executed with the `mysql_stmt_prepare()` and `mysql_stmt_execute()` C API functions, the server writes `Prepare` and `Execute` lines to the general query log so that you can tell when statements are prepared and executed.

Suppose that you prepare and execute a statement as follows:

1. Call `mysql_stmt_prepare()` to prepare the statement string "SELECT ?".
2. Call `mysql_stmt_bind_param()` to bind the value 3 to the parameter in the prepared statement.
3. Call `mysql_stmt_execute()` to execute the prepared statement.

As a result of the preceding calls, the server writes the following lines to the general query log:

```
Prepare [1] SELECT ?  
Execute [1] SELECT 3
```

Each Prepare and Execute line in the log is tagged with a [N] statement identifier so that you can keep track of which prepared statement is being logged. *N* is a positive integer. If there are multiple prepared statements active simultaneously for the client, *N* may be greater than 1. Each Execute lines shows a prepared statement after substitution of data values for ? parameters.

Version notes: Prepare lines are displayed without [N] before MySQL 4.1.10. Execute lines are not displayed at all before MySQL 4.1.10.

22.2.7. C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions described in detail in the following sections.

Note that all functions operating with a MYSQL_STMT structure begin with the prefix `mysql_stmt_`.

To create a MYSQL_STMT handle, use the `mysql_stmt_init()` function.

22.2.7.1. `mysql_stmt_affected_rows()`

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

Returns the total number of rows changed, deleted, or inserted by the last executed statement. May be called immediately after `mysql_stmt_execute()` for UPDATE, DELETE, or INSERT statements. For SELECT statements, `mysql_stmt_affected_rows()` works like `mysql_num_rows()`.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an UPDATE statement, no rows matched the WHERE clause in the query, or that no query has yet been executed. -1 indicates that the query returned an error or that, for a SELECT query, `mysql_stmt_affected_rows()` was called prior to calling

`mysql_stmt_store_result()`. Because `mysql_stmt_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

See [Section 22.2.3.1, “mysql_affected_rows\(\)”](#), for additional information on the return value.

Errors

None.

Example

For the usage of `mysql_stmt_affected_rows()`, refer to the Example from [Section 22.2.7.10, “mysql_stmt_execute\(\)”](#).

22.2.7.2. `mysql_stmt_attr_get()`

```
int mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type
option, void *arg)
```

Description

Can be used to get the current value for a statement attribute.

The `option` argument is the option that you want to get; the `arg` should point to a variable that should contain the option value. If the option is an integer, then `arg` should point to the value of the integer.

See [Section 22.2.7.3, “mysql_stmt_attr_set\(\)”](#), for a list of options and option types.

Note: In MySQL 5.0, `mysql_stmt_attr_get()` uses `unsigned int *`, not `my_bool *`, for `STMT_ATTR_UPDATE_MAX_LENGTH`. This is corrected in MySQL 5.1.7.

Return Values

0 if okay. Non-zero if option is unknown.

Errors

None.

22.2.7.3. `mysql_stmt_attr_set()`

```
int mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type
option, const void *arg)
```

Description

Can be used to affect behavior for a prepared statement. This function may be called multiple times to set several options.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, then `arg` should point to the value of the integer.

Possible option values:

Option	Argument Type	Function
STMT_ATTR_UPDATE_MAX_LENGTH	my_bool *	If set to 1: Update metadata MySQL_FIELD->max_length in mysql_stmt_store_result().
STMT_ATTR_CURSOR_TYPE	unsigned long *	Type of cursor to open for statement when mysql_stmt_execute() is invoked. *arg can be CURSOR_TYPE_NO_CURSOR (the default) or CURSOR_TYPE_READ_ONLY.
STMT_ATTR_PREFETCH_ROWS	unsigned long *	Number of rows to fetch from server at a time when using a cursor. *arg can be in the range from 1 to the maximum value of unsigned long. The default is 1.

Note: In MySQL 5.0, `mysql_stmt_attr_get()` uses `unsigned int *`, not `my_bool *`, for `STMT_ATTR_UPDATE_MAX_LENGTH`. This is corrected in MySQL 5.1.7.

If you use the `STMT_ATTR_CURSOR_TYPE` option with `CURSOR_TYPE_READ_ONLY`, a cursor is opened for the statement when you invoke `mysql_stmt_execute()`. If there is already an open cursor from a previous `mysql_stmt_execute()` call, it closes the cursor before opening a new one. `mysql_stmt_reset()` also closes any open cursor before preparing the statement for re-execution. `mysql_stmt_free_result()` closes any open cursor.

If you open a cursor for a prepared statement, `mysql_stmt_store_result()` is unnecessary, because that function causes the result set to be buffered on the client side.

The `STMT_ATTR_CURSOR_TYPE` option was added in MySQL 5.0.2. The `STMT_ATTR_PREFETCH_ROWS` option was added in MySQL 5.0.6.

Return Values

0 if okay. Non-zero if option is unknown.

Errors

None.

Example

The following example opens a cursor for a prepared statement and sets the number of rows to fetch at a time to 5:

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type)
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                          (void*) &prefetch_rows);
/* ... check return value ... */
```

22.2.7.4. `mysql_stmt_bind_param()`

```
my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_param()` is used to bind data for the parameter markers in the SQL statement that was passed to `mysql_stmt_prepare()`. It uses `MYSQL_BIND` structures to supply the data. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain an element for each ‘?’ parameter marker that is present in the query.

Suppose that you prepare the following statement:

```
INSERT INTO mytbl VALUES(?, ?, ?)
```

When you bind the parameters, the array of `MYSQL_BIND` structures must contain three elements, and can be declared like this:

```
MYSQL_BIND bind[3];
```

The members of each `MYSQL_BIND` element that should be set are described in [Section 22.2.5, “C API Prepared Statement Data types”](#).

Return Values

Zero if the bind was successful. Non-zero if an error occurred.

Errors

- `CR_INVALID_BUFFER_USE`

Indicates if the bind is to supply the long data in chunks and if the buffer type is non string or binary.

- `CR_UNSUPPORTED_PARAM_TYPE`

The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

Out of memory.

- CR_UNKNOWN_ERROR

An unknown error occurred.

Example

For the usage of `mysql_stmt_bind_param()`, refer to the Example from [Section 22.2.7.10, “mysql_stmt_execute\(\)”](#).

22.2.7.5. `mysql_stmt_bind_result()`

```
my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_result()` is used to associate (bind) columns in the result set to data buffers and length buffers. When `mysql_stmt_fetch()` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified buffers.

All columns must be bound to buffers prior to calling `mysql_stmt_fetch()`. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain an element for each column of the result set. If you do not bind columns to `MYSQL_BIND` structures, `mysql_stmt_fetch()` simply ignores the data fetch. The buffers should be large enough to hold the data values, because the protocol doesn't return data values in chunks.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysql_stmt_fetch()` is called. Suppose that an application binds the columns in a result set and calls `mysql_stmt_fetch()`. The client/server protocol returns data in the bound buffers. Then suppose that the application binds the columns to a different set of buffers. The protocol does not place data into the newly bound buffers until the next call to `mysql_stmt_fetch()` occurs.

To bind a column, an application calls `mysql_stmt_bind_result()` and passes the type, address, and the address of the length buffer. The members of each `MYSQL_BIND` element that should be set are described in [Section 22.2.5, “C API](#)

[Prepared Statement Data types](#)".

Return Values

Zero if the bind was successful. Non-zero if an error occurred.

Errors

- CR_UNSUPPORTED_PARAM_TYPE

The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- CR_OUT_OF_MEMORY

Out of memory.

- CR_UNKNOWN_ERROR

An unknown error occurred.

Example

For the usage of `mysql_stmt_bind_result()`, refer to the Example from [Section 22.2.7.11, "mysql_stmt_fetch\(\)"](#).

22.2.7.6. `mysql_stmt_close()`

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

Description

Closes the prepared statement. `mysql_stmt_close()` also deallocates the statement handle pointed to by `stmt`.

If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Return Values

Zero if the statement was freed successfully. Non-zero if an error occurred.

Errors

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_UNKNOWN_ERROR

An unknown error occurred.

Example

For the usage of `mysql_stmt_close()`, refer to the Example from [Section 22.2.7.10, “mysql_stmt_execute\(\)”](#).

22.2.7.7. `mysql_stmt_data_seek()`

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a statement result set. The `offset` value is a row number and should be in the range from 0 to `mysql_stmt_num_rows(stmt) - 1`.

This function requires that the statement result set structure contains the entire result of the last executed query, so `mysql_stmt_data_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

None.

Errors

None.

22.2.7.8. `mysql_stmt_errno()`

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_errno()` returns the error code for the most recently invoked statement API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysql_error.h`. Errors also are listed at [Appendix B, Error Codes and Messages](#).

Return Values

An error code value. Zero if no error occurred.

Errors

None.

22.2.7.9. `mysql_stmt_error()`

```
const char *mysql_stmt_error(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_error()` returns a null-terminated string containing the error message for the most recently invoked statement API function that can succeed or fail. An empty string ("") is returned if no error occurred. This means the following two tests are equivalent:

```
if (mysql_stmt_errno(stmt))
{
    // an error occurred
}

if (mysql_stmt_error(stmt)[0])
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

22.2.7.10. `mysql_stmt_execute()`

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_execute()` executes the prepared query associated with the statement handle. The currently bound parameter marker values are sent to server during this call, and the server replaces the markers with this newly supplied data.

If the statement is an UPDATE, DELETE, or INSERT, the total number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`. If this is a statement such as SELECT that generates a result set, you must call `mysql_stmt_fetch()` to fetch the data prior to calling any other functions that result in query processing. For more information on how to fetch the results, refer to [Section 22.2.7.11, “mysql_stmt_fetch\(\)”](#).

For statements that generate a result set, you can request that `mysql_stmt_execute()` open a cursor for the statement by calling `mysql_stmt_attr_set()` before executing the statement. If you execute a statement multiple times, `mysql_stmt_execute()` closes any open cursor before opening a new one.

Return Values

Zero if execution was successful. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_OUT_OF_MEMORY

Out of memory.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

Example

The following example demonstrates how to create and populate a table using `mysql_stmt_init()`, `mysql_stmt_prepare()`, `mysql_stmt_param_count()`, `mysql_stmt_bind_param()`, `mysql_stmt_execute()`, and `mysql_stmt_affected_rows()`. The `mysql` variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                                    col2 VARCHAR(40),\
                                                    col3 SMALLINT,\
                                                    col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO test_table(col1,col2,col3) VALUES

MYSQL_STMT      *stmt;
MYSQL_BIND      bind[3];
my_ulonglong    affected_rows;
int             param_count;
short           small_data;
int             int_data;
char            str_data[STRING_SIZE];
unsigned long   str_length;
my_bool         is_null;
```

```

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
    fprintf(stderr, " DROP TABLE failed\n");
    fprintf(stderr, " %s\n", mysql_error(mysql));
    exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
    fprintf(stderr, " CREATE TABLE failed\n");
    fprintf(stderr, " %s\n", mysql_error(mysql));
    exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/* sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
    fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
    fprintf(stderr, " invalid parameter count returned by MySQL\n");
    exit(0);
}

/* Bind the data for all 3 parameters */

memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need to specify buffer_leng
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;

```

```

bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
    fprintf(stderr, " mysql_stmt_bind_param() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Specify the data values for the first row */
int_data= 10;          /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get the total number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
        (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, " invalid affected rows by MySQL\n");
    exit(0);
}

```

```

/* Specify data values for second row, then re-execute the statement
int_data= 1000;
strncpy(str_data, "The most popular Open Source database", STRING_SI
str_length= strlen(str_data);
small_data= 1000;          /* smallint */
is_null= 0;                /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
        (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
    fprintf(stderr, " invalid affected rows by MySQL\n");
    exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
    fprintf(stderr, " failed while closing the statement\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

```

Note: For complete examples on the use of prepared statement functions, refer to the file `tests/mysql_client_test.c`. This file can be obtained from a MySQL source distribution or from the BitKeeper source repository.

22.2.7.11. `mysql_stmt_fetch()`

```
int mysql_stmt_fetch(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_fetch()` returns the next row in the result set. It can be called only

while the result set exists; that is, after a call to `mysql_stmt_execute()` that creates a result set or after `mysql_stmt_store_result()`, which is called after `mysql_stmt_execute()` to buffer the entire result set.

`mysql_stmt_fetch()` returns row data using the buffers bound by `mysql_stmt_bind_result()`. It returns the data in those buffers for all the columns in the current row set and the lengths are returned to the length pointer.

All columns must be bound by the application before calling `mysql_stmt_fetch()`.

If a fetched data value is a NULL value, the `*is_null` value of the corresponding `MYSQL_BIND` structure contains TRUE (1). Otherwise, the data and its length are returned in the `*buffer` and `*length` elements based on the buffer type specified by the application. Each numeric and temporal type has a fixed length, as listed in the following table. The length of the string types depends on the length of the actual data value, as indicated by `data_length`.

Type	Length
MYSQL_TYPE_TINY	1
MYSQL_TYPE_SHORT	2
MYSQL_TYPE_LONG	4
MYSQL_TYPE_LONGLONG	8
MYSQL_TYPE_FLOAT	4
MYSQL_TYPE_DOUBLE	8
MYSQL_TYPE_TIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATE	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATETIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_STRING	data length
MYSQL_TYPE_BLOB	data_length

Return Values

Return Value	Description
0	Successful, the data has been fetched to application data buffers.

1	Error occurred. Error code and message can be obtained by calling <code>mysql_stmt_errno()</code> and <code>mysql_stmt_error()</code> .
<code>MYSQL_NO_DATA</code>	No more rows/data exists
<code>MYSQL_DATA_TRUNCATED</code>	Data truncation occurred

`MYSQL_DATA_TRUNCATED` is returned when truncation reporting is enabled. (Reporting is enabled by default, but can be controlled with `mysql_options()`.) To determine which parameters were truncated when this value is returned, check the error members of the `MYSQL_BIND` parameter structures.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

- `CR_UNSUPPORTED_PARAM_TYPE`

The buffer type is `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, but the data type is not `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP`.

- All other unsupported conversion errors are returned from

```
mysql_stmt_bind_result().
```

Example

The following example demonstrates how to fetch data from a table using `mysql_stmt_result_metadata()`, `mysql_stmt_bind_result()`, and `mysql_stmt_fetch()`. (This example expects to retrieve the two rows inserted by the example shown in [Section 22.2.7.10, “mysql_stmt_execute\(\)”](#).) The `mysql` variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 FROM test_table

MYSQL_STMT      *stmt;
MYSQL_BIND      bind[4];
MYSQL_RES       *prepare_meta_result;
MYSQL_TIME      ts;
unsigned long   length[4];
int             param_count, column_count, row_count;
short          small_data;
int            int_data;
char           str_data[STRING_SIZE];
my_bool        is_null[4];
my_bool        error[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
    fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
```

```

    fprintf(stderr, " invalid parameter count returned by MySQL\n");
    exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
    fprintf(stderr,
            " mysql_stmt_result_metadata(), returned no meta informatio
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout, " total columns in SELECT statement: %d\n", column_c

if (column_count != 4) /* validate column count */
{
    fprintf(stderr, " invalid column count returned by MySQL\n");
    exit(0);
}

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, " mysql_stmt_execute(), failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */

memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];
bind[0].error= &error[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];
bind[1].error= &error[1];

```

```

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];
bind[2].error= &error[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];
bind[3].error= &error[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
    fprintf(stderr, " mysql_stmt_bind_result() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Now buffer all results to client */
if (mysql_stmt_store_result(stmt))
{
    fprintf(stderr, " mysql_stmt_store_result() failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ... \n");
while (!mysql_stmt_fetch(stmt))
{
    row_count++;
    fprintf(stdout, " row %d\n", row_count);

    /* column 1 */
    fprintf(stdout, " column1 (integer) : ");
    if (is_null[0])
        fprintf(stdout, " NULL\n");
    else
        fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

    /* column 2 */
    fprintf(stdout, " column2 (string) : ");
    if (is_null[1])
        fprintf(stdout, " NULL\n");
}

```

```

else
    fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

/* column 3 */
fprintf(stdout, "    column3 (smallint) : ");
if (is_null[2])
    fprintf(stdout, " NULL\n");
else
    fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

/* column 4 */
fprintf(stdout, "    column4 (timestamp): ");
if (is_null[3])
    fprintf(stdout, " NULL\n");
else
    fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
            ts.year, ts.month, ts.day,
            ts.hour, ts.minute, ts.second,
            length[3]);
fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
    fprintf(stderr, " MySQL failed to return all rows\n");
    exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);

/* Close the statement */
if (mysql_stmt_close(stmt))
{
    fprintf(stderr, " failed while closing the statement\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
}

```

22.2.7.12. `mysql_stmt_fetch_column()`

```

int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind,
unsigned int column, unsigned long offset)

```

Description

Fetch one column from the current result set row. `bind` provides the buffer where data should be placed. It should be set up the same way as for `mysql_stmt_bind_result()`. `column` indicates which column to fetch. The first column is numbered 0. `offset` is the offset within the data value at which to begin retrieving data. This can be used for fetching the data value in pieces. The beginning of the value is offset 0.

Return Values

Zero if the value was fetched successfully. Non-zero if an error occurred.

Errors

- `CR_INVALID_PARAMETER_NO`

Invalid column number.

- `CR_NO_DATA`

The end of the result set has already been reached.

22.2.7.13. `mysql_stmt_field_count()`

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

Description

Returns the number of columns for the most recent statement for the statement handler. This value is zero for statements such as `INSERT` or `DELETE` that do not produce result sets.

`mysql_stmt_field_count()` can be called after you have prepared a statement by invoking `mysql_stmt_prepare()`.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

22.2.7.14. `mysql_stmt_free_result()`

```
my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

Description

Releases memory associated with the result set produced by execution of the prepared statement. If there is a cursor open for the statement, `mysql_stmt_free_result()` closes it.

Return Values

Zero if the result set was freed successfully. Non-zero if an error occurred.

Errors

22.2.7.15. `mysql_stmt_init()`

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

Description

Create a `MYSQL_STMT` handle. The handle should be freed with `mysql_stmt_close(MYSQL_STMT *)`.

Return values

A pointer to a `MYSQL_STMT` structure in case of success. `NULL` if out of memory.

Errors

- `CR_OUT_OF_MEMORY`

Out of memory.

22.2.7.16. `mysql_stmt_insert_id()`

```
my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the prepared `INSERT` or `UPDATE` statement. Use this function after you have executed a prepared `INSERT` statement on a table which contains an `AUTO_INCREMENT` field.

See [Section 22.2.3.36, “mysql_insert_id\(\)”](#), for more information.

Return Values

Value for `AUTO_INCREMENT` column which was automatically generated or explicitly set during execution of prepared statement, or value generated by `LAST_INSERT_ID(expr)` function. Return value is undefined if statement does not set `AUTO_INCREMENT` value.

Errors

None.

22.2.7.17. `mysql_stmt_num_rows()`

```
my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

Description

Returns the number of rows in the result set.

The use of `mysql_stmt_num_rows()` depends on whether you used `mysql_stmt_store_result()` to buffer the entire result set in the statement handle.

If you use `mysql_stmt_store_result()`, `mysql_stmt_num_rows()` may be called immediately.

Return Values

The number of rows in the result set.

Errors

None.

22.2.7.18. `mysql_stmt_param_count()`

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

Description

Returns the number of parameter markers present in the prepared statement.

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

For the usage of `mysql_stmt_param_count()`, refer to the Example from [Section 22.2.7.10, “`mysql_stmt_execute\(\)`”](#).

22.2.7.19. `mysql_stmt_param_metadata()`

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

This function currently does nothing.

Description

Return Values

Errors

22.2.7.20. `mysql_stmt_prepare()`

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query,  
unsigned long length)
```

Description

Given the statement handle returned by `mysql_stmt_init()`, prepares the SQL statement pointed to by the string `query` and returns a status value. The string length should be given by the `length` argument. The string must consist of a single SQL statement. You should not add a terminating semicolon (`;`) or `\g` to the statement.

The application can include one or more parameter markers in the SQL statement by embedding question mark (`?`) characters into the SQL string at the appropriate positions.

The markers are legal only in certain places in SQL statements. For example, they are allowed in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value. However, they are not allowed for identifiers (such as table or column names), or to specify both operands of a binary operator such as the = equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

The parameter markers must be bound to application variables using `mysql_stmt_bind_param()` before executing the statement.

Return Values

Zero if the statement was prepared successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query

- CR_UNKNOWN_ERROR

An unknown error occurred.

If the prepare operation was unsuccessful (that is, `mysql_stmt_prepare()` returns non-zero), the error message can be obtained by calling `mysql_stmt_error()`.

Example

For the usage of `mysql_stmt_prepare()`, refer to the Example from [Section 22.2.7.10, “mysql_stmt_execute\(\)”](#).

22.2.7.21. mysql_stmt_reset()

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Reset the prepared statement on the client and server to state after prepare. This is mainly used to reset data sent with `mysql_stmt_send_long_data()`. Any open cursor for the statement is closed.

To re-prepare the statement with another query, use `mysql_stmt_prepare()`.

Return Values

Zero if the statement was reset successfully. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query

- CR_UNKNOWN_ERROR

An unknown error occurred.

22.2.7.22. `mysql_stmt_result_metadata()`

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```

Description

If a statement passed to `mysql_stmt_prepare()` is one that produces a result set, `mysql_stmt_result_metadata()` returns the result set metadata in the form of a pointer to a `MYSQL_RES` structure that can be used to process the meta information such as total number of fields and individual field information. This result set pointer can be passed as an argument to any of the field-based API functions that process result set metadata, such as:

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`
- `mysql_field_seek()`
- `mysql_field_tell()`
- `mysql_free_result()`

The result set structure should be freed when you are done with it, which you can do by passing it to `mysql_free_result()`. This is similar to the way you free a result set obtained from a call to `mysql_store_result()`.

The result set returned by `mysql_stmt_result_metadata()` contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with `mysql_stmt_fetch()`.

Return Values

A `MYSQL_RES` result structure. `NULL` if no meta information exists for the prepared query.

Errors

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

Example

For the usage of `mysql_stmt_result_metadata()`, refer to the Example from [Section 22.2.7.11, “mysql_stmt_fetch\(\)”](#).

22.2.7.23. `mysql_stmt_row_seek()`

```
MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt,  
MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a statement result set. The `offset` value is a row offset that should be a value returned from `mysql_stmt_row_tell()` or from `mysql_stmt_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_stmt_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_stmt_row_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_stmt_row_seek()`.

Errors

None.

22.2.7.24. `mysql_stmt_row_tell()`

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

Description

Returns the current position of the row cursor for the last `mysql_stmt_fetch()`. This value can be used as an argument to `mysql_stmt_row_seek()`.

You should use `mysql_stmt_row_tell()` only after `mysql_stmt_store_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

22.2.7.25. `mysql_stmt_send_long_data()`

```
my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int  
parameter_number, const char *data, unsigned long length)
```

Description

Allows an application to send parameter data to the server in pieces (or “chunks”). This function can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the TEXT or BLOB data types.

`parameter_number` indicates which parameter to associate the data with. Parameters are numbered beginning with 0. `data` is a pointer to a buffer containing data to be sent, and `length` indicates the number of bytes in the buffer.

Note: The next `mysql_stmt_execute()` call ignores the bind buffer for all parameters that have been used with `mysql_stmt_send_long_data()` since last `mysql_stmt_execute()` or `mysql_stmt_reset()`.

If you want to reset/forget the sent data, you can do it with `mysql_stmt_reset()`. See [Section 22.2.7.21, “mysql_stmt_reset\(\)”](#).

Return Values

Zero if the data is sent successfully to server. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

The MySQL server has gone away.

- CR_OUT_OF_MEMORY

Out of memory.

- CR_UNKNOWN_ERROR

An unknown error occurred.

Example

The following example demonstrates how to send the data for a TEXT column in chunks. It inserts the data value 'MySQL - The most popular Open Source database' into the text_column column. The mysql variable is assumed to be a valid connection handle.

```
#define INSERT_QUERY "INSERT INTO test_long_data(text_column) VALUES

MYSQL_BIND bind[1];
long      length;

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}
memset(bind, 0, sizeof(bind));
bind[0].buffer_type= MYSQL_TYPE_STRING;
bind[0].length= &length;
bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
    fprintf(stderr, "\n param bind failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Supply data in chunks to server */
if (!mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
    fprintf(stderr, "\n send_long_data failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* Supply the next piece of data */
if (mysql_stmt_send_long_data(stmt,0," - The most popular Open Sour
{
    fprintf(stderr, "\n send_long_data failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
```

```

    exit(0);
}

/* Now, execute the query */
if (mysql_stmt_execute(stmt))
{
    fprintf(stderr, "\n mysql_stmt_execute failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

```

22.2.7.26. `mysql_stmt_sqlstate()`

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_sqlstate()` returns a null-terminated string containing the SQLSTATE error code for the most recently invoked prepared statement API function that can succeed or fail. The error code consists of five characters. "00000" means "no error." The values are specified by ANSI SQL and ODBC. For a list of possible values, see [Appendix B, Error Codes and Messages](#).

Note that not all MySQL errors are yet mapped to SQLSTATE codes. The value "HY000" (general error) is used for unmapped errors.

Return Values

A null-terminated character string containing the SQLSTATE error code.

22.2.7.27. `mysql_stmt_store_result()`

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Description

You must call `mysql_stmt_store_result()` for every statement that successfully produces a result set (SELECT, SHOW, DESCRIBE, EXPLAIN), and only if you want to buffer the complete result set by the client, so that the subsequent `mysql_stmt_fetch()` call returns buffered data.

It is unnecessary to call `mysql_stmt_store_result()` for other statements, but if you do, it does not harm or cause any notable performance problem. You can detect whether the statement produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to [Section 22.2.7.22, “mysql_stmt_result_metadata\(\)”](#).

Note: MySQL doesn't by default calculate `MYSQL_FIELD->max_length` for all columns in `mysql_stmt_store_result()` because calculating this would slow down `mysql_stmt_store_result()` considerably and most applications doesn't need `max_length`. If you want `max_length` to be updated, you can call `mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)` to enable this. See [Section 22.2.7.3, “mysql_stmt_attr_set\(\)”](#).

Return Values

Zero if the results are buffered successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

Out of memory.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

22.2.8. C API Prepared statement problems

Here follows a list of the currently known problems with prepared statements:

- TIME, TIMESTAMP, and DATETIME do not support parts of seconds (for example from DATE_FORMAT()).
- When converting an integer to string, ZEROFILL is honored with prepared statements in some cases where the MySQL server doesn't print the leading zeros. (For example, with MIN(number-with-zerofill)).
- When converting a floating point number to a string in the client, the rightmost digits of the converted value may differ slightly from those of the original value.
- *Prepared statements do not use the Query Cache, even in cases where a query does not contain any placeholders.* See [Section 5.14.1, “How the Query Cache Operates”](#).
- Prepared statements do not support multi-statements (that is, multiple statements within a single string separated by ‘;’ characters). This also means that prepared statements cannot invoke stored procedures that return result sets, because prepared statements do not support multiple result sets.

22.2.9. C API Handling of Multiple Statement Execution

MySQL 5.0 supports the execution of multiple statements specified in a single query string. To use this capability with a given connection, you must specify the CLIENT_MULTI_STATEMENTS option in the flags parameter to mysql_real_connect() when opening the connection. You can also set this for an existing connection by calling mysql_set_server_option(MYSQL_OPTION_MULTI_STATEMENTS_ON).

By default, mysql_query() and mysql_real_query() return only the first query status and the subsequent queries status can be processed using mysql_more_results() and mysql_next_result().

If you enable multiple-statement support, you should process the results from mysql_query() and mysql_real_query() within a loop that checks for more results. This is true even for statements such as DROP TABLE that return a result but not a result set. Failure to process the result this way may result in a dropped

connection to the server.

```
/* Connect to server with option CLIENT_MULTI_STATEMENTS */
mysql_real_connect(..., CLIENT_MULTI_STATEMENTS);

/* Now execute multiple queries */
mysql_query(mysql, "DROP TABLE IF EXISTS test_table;\
                  CREATE TABLE test_table(id INT);\
                  INSERT INTO test_table VALUES(10);\
                  UPDATE test_table SET id=20 WHERE id=10;\
                  SELECT * FROM test_table;\
                  DROP TABLE test_table");

do
{
    /* Process all results */
    ...
    printf("total affected rows: %lld", mysql_affected_rows(mysql));
    ...
    if (!(result= mysql_store_result(mysql)))
    {
        printf(stderr, "Got fatal error processing query\n");
        exit(1);
    }
    process_result_set(result); /* client function */
    mysql_free_result(result);
} while (!mysql_next_result(mysql));
```

The multiple-statement capability can be used with `mysql_query()` or `mysql_real_query()`. It cannot be used with the prepared statement interface. Prepared statement handles are defined to work only with strings that contain a single statement.

22.2.10. C API Handling of Date and Time Values

The binary protocol allows you to send and receive date and time values (DATE, TIME, DATETIME, and TIMESTAMP), using the `MYSQL_TIME` structure. The members of this structure are described in [Section 22.2.5, “C API Prepared Statement Data types”](#).

To send temporal data values, create a prepared statement using `mysql_stmt_prepare()`. Then, before calling `mysql_stmt_execute()` to execute the statement, use the following procedure to set up each temporal parameter:

1. In the `MYSQL_BIND` structure associated with the data value, set the

buffer_type member to the type that indicates what kind of temporal value you're sending. For DATE, TIME, DATETIME, or TIMESTAMP values, set buffer_type to MYSQL_TYPE_DATE, MYSQL_TYPE_TIME, MYSQL_TYPE_DATETIME, or MYSQL_TYPE_TIMESTAMP, respectively.

2. Set the buffer member of the MYSQL_BIND structure to the address of the MYSQL_TIME structure in which you pass the temporal value.
3. Fill in the members of the MYSQL_TIME structure that are appropriate for the type of temporal value to be passed.

Use `mysql_stmt_bind_param()` to bind the parameter data to the statement. Then you can call `mysql_stmt_execute()`.

To retrieve temporal values, the procedure is similar, except that you set the `buffer_type` member to the type of value you expect to receive, and the `buffer` member to the address of a `MYSQL_TIME` structure into which the returned value should be placed. Use `mysql_bind_results()` to bind the buffers to the statement after calling `mysql_stmt_execute()` and before fetching the results.

Here is a simple example that inserts DATE, TIME, and TIMESTAMP data. The `mysql` variable is assumed to be a valid connection handle.

```
MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field,
                                         timestamp_field) VALUES(?,?,

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
    fprintf(stderr, " mysql_stmt_init(), out of memory\n");
    exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
    fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
    fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
    exit(0);
}

/* set up input buffers for all 3 parameters */
```

```

bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
bind[1]= bind[2]= bind[0];
...

mysql_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..

```

22.2.11. C API Threaded Function Descriptions

You need to use the following functions when you want to create a threaded client. See [Section 22.2.15, “How to Make a Threaded Client”](#).

22.2.11.1. my_init()

```
void my_init(void)
```

Description

This function needs to be called once in the program before calling any MySQL function. This initializes some global variables that MySQL needs. If you are using a thread-safe client library, this also calls `mysql_thread_init()` for this thread.

This is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()` and `mysql_connect()`.

Return Values

None.

22.2.11.2. mysql_thread_init()

```
my_bool mysql_thread_init(void)
```

Description

This function needs to be called for each created thread to initialize thread-specific variables.

This is automatically called by `my_init()` and `mysql_connect()`.

Return Values

Zero if successful. Non-zero if an error occurred.

22.2.11.3. mysql_thread_end()

```
void mysql_thread_end(void)
```

Description

This function needs to be called before calling `pthread_exit()` to free memory allocated by `mysql_thread_init()`.

Note that this function *is not invoked automatically by the client library*. It must be called explicitly to avoid a memory leak.

Return Values

None.

22.2.11.4. mysql_thread_safe()

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client is compiled as thread-safe.

Return Values

1 if the client is thread-safe, 0 otherwise.

22.2.12. C API Embedded Server Function Descriptions

If you want to allow your application to be linked against the embedded MySQL server library, you must use the `mysql_server_init()` and `mysql_server_end()` functions. See [Section 22.1, “libmysqld, the Embedded MySQL Server Library”](#).

However, to provide improved memory management, even programs that are linked with `-lmysqlclient` rather than `-lmysqld` should include calls to begin and end use of the library. As of MySQL 5.0.3, the `mysql_library_init()` and `mysql_library_end()` functions can be used to do this. These actually are `#define` symbols that make them equivalent to `mysql_server_init()` and `mysql_server_end()`, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL C API library no matter whether the application uses `libmysqlclient` or `libmysqld`. For more information, see [Section 22.2.2, “C API Function Overview”](#).

22.2.12.1. `mysql_server_init()`

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function **must** be called once in the program using the embedded server before calling any other MySQL function. It starts the server and initializes any subsystems (`mysys`, `InnoDB`, and so forth) that the server uses. If this function is not called, the next call to `mysql_init()` executes `mysql_server_init()`.

In a non-multi-threaded environment, the call to `mysql_server_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, a race condition is possible if `mysql_server_init()` is invoked by `mysql_init()` in a multi-threaded environment: `mysql_server_init()` is not thread-safe, so it should be called prior to any other client library call.

If you are using the `DEBUG` package that comes with MySQL, you should call `mysql_server_init()` after you have called `my_init()`.

The `argc` and `argv` arguments are analogous to the arguments to `main()`. The first element of `argv` is ignored (it typically contains the program name). For convenience, `argc` may be 0 (zero) if there are no command-line arguments for the server. `mysql_server_init()` makes a copy of the arguments so it's safe to destroy `argv` or groups after the call.

If you want to connect to an external server without starting the embedded server, you have to specify a negative value for `argc`.

The NULL-terminated list of strings in `groups` selects which groups in the option files are active. See [Section 4.3.2, “Using Option Files”](#). For convenience, groups may be NULL, in which case the `[server]` and `[embedded]` groups are active.

Example

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
    "this_program",          /* this string is not used */
    "--datadir=.",
    "--key_buffer_size=32M"
};
static char *server_groups[] = {
    "embedded",
    "server",
    "this_program_SERVER",
    (char *)NULL
};

int main(void) {
    if (mysql_server_init(sizeof(server_args) / sizeof(char *),
                          server_args, server_groups))
        exit(1);

    /* Use any MySQL API functions here */

    mysql_server_end();

    return EXIT_SUCCESS;
}
```

Return Values

0 if okay, 1 if an error occurred.

22.2.12.2. `mysql_server_end()`

```
void mysql_server_end(void)
```

Description

This function **must** be called once in the program after all other MySQL functions. It shuts down the embedded server.

Return Values

None.

22.2.13. Common Questions and Problems When Using the C API

22.2.13.1. Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).
- The data couldn't be read (an error occurred on the connection).
- The query returned no data (for example, it was an `INSERT`, `UPDATE`, or `DELETE`).

You can always check whether the statement should have produced a non-empty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an `INSERT` or a `DELETE`). If `mysql_field_count()` returns a non-zero value, the statement should have produced a non-empty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

22.2.13.2. What Results You Can Get from a Query

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an INSERT, UPDATE, or DELETE.

For a fast re-create, use `TRUNCATE TABLE`.

- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With `mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.
- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an `AUTO_INCREMENT` index. See [Section 22.2.3.36, “mysql_insert_id\(\)”](#).
- Some queries (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) return additional information. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

22.2.13.3. How to Get the Unique ID for the Last Inserted Row

If you insert a record into a table that contains an `AUTO_INCREMENT` column, you can obtain the value stored into that column by calling the `mysql_insert_id()` function.

You can check from your C applications whether a value was stored in an `AUTO_INCREMENT` column by executing the following code (which assumes that you've checked that the statement succeeded). It determines whether the query was an INSERT with an `AUTO_INCREMENT` index:

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
```

```
{
    used_id = mysql_insert_id(&mysql);
}
```

For more information, see [Section 22.2.3.36, “mysql_insert_id\(\)”](#).

When a new `AUTO_INCREMENT` value has been generated, you can also obtain it by executing a `SELECT LAST_INSERT_ID()` statement with `mysql_query()` and retrieving the value from the result set returned by the statement.

For `LAST_INSERT_ID()`, the most recently generated ID is maintained in the server on a per-connection basis. It is not changed by another client. It is not even changed if you update another `AUTO_INCREMENT` column with a non-magic value (that is, a value that is not `NULL` and not `0`).

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto, text)
    VALUES(NULL, 'text');           # generate ID by inserting NUL
INSERT INTO foo2 (id, text)
    VALUES(LAST_INSERT_ID(), 'text'); # use ID in second table
```

Note that `mysql_insert_id()` returns the value stored into an `AUTO_INCREMENT` column, whether that value is automatically generated by storing `NULL` or `0` or was specified as an explicit value. `LAST_INSERT_ID()` returns only automatically generated `AUTO_INCREMENT` values. If you store an explicit value other than `NULL` or `0`, it does not affect the value returned by `LAST_INSERT_ID()`.

22.2.13.4. Problems Linking with the C API

When linking with the C API, the following errors may occur on some systems:

```
gcc -g -o client test.o -L/usr/local/lib/mysql -lmysqlclient -lsocle
Undefined          first referenced
  symbol           in file
floor              /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client
```

If this happens on your system, you must include the math library by adding `-lm` to the end of the compile/link line.

22.2.14. Building Client Programs

If you compile MySQL clients that you've written yourself or that you obtain from a third-party, they must be linked using the `-lmysqlclient -lz` options in the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in `/usr/local/mysql/lib`, use `-L/usr/local/mysql/lib -lmysqlclient -lz` in the link command.

For clients that use MySQL header files, you may need to specify an `-I` option when you compile them (for example, `-I/usr/local/mysql/include`), so that the compiler can find the header files.

To make it simpler to compile MySQL programs on Unix, we have provided the **mysql_config** script for you. See [Section 22.9.2, “mysql_config — Get Compile Options for Compiling Clients”](#).

You can use it to compile a MySQL client as follows:

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname ` $CFG --cflags ` progname.c ` $CFG --libs `"
```

The `sh -c` is needed to get the shell not to treat the output from **mysql_config** as one word.

22.2.15. How to Make a Threaded Client

The client library is almost thread-safe. The biggest problem is that the subroutines in `net.c` that read from sockets are not interrupt safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server. If you install interrupt handlers for the `SIGPIPE` interrupt, the socket handling should be thread-safe.

To avoid aborting the program when a connection terminates, MySQL blocks `SIGPIPE` on the first call to `mysql_server_init()`, `mysql_init()`, or `mysql_connect()`. If you want to use your own `SIGPIPE` handler, you should first call `mysql_server_init()` and then install your handler.

In the older binaries we distribute on our Web site (<http://www.mysql.com/>), the client libraries are not normally compiled with the thread-safe option (the

Windows binaries are by default compiled to be thread-safe). Newer binary distributions should have both a normal and a thread-safe client library.

To get a threaded client where you can interrupt the client from other threads and set timeouts when talking with the MySQL server, you should use the `-lmysys`, `-lmystrings`, and `-ldebug` libraries and the `net_serv.o` code that the server uses.

If you don't need interrupts or timeouts, you can just compile a thread-safe client library (`mysqlclient_r`) and use this. See [Section 22.2, “MySQL C API”](#). In this case, you don't have to worry about the `net_serv.o` object file or the other MySQL libraries.

When using a threaded client and you want to use timeouts and interrupts, you can make great use of the routines in the `thr_alarm.c` file. If you are using routines from the `mysys` library, the only thing you must remember is to call `my_init()` first! See [Section 22.2.11, “C API Threaded Function Descriptions”](#).

All functions except `mysql_real_connect()` are by default thread-safe. The following notes describe how to compile a thread-safe client library and use it in a thread-safe manner. (The notes below for `mysql_real_connect()` actually apply to `mysql_connect()` as well, but because `mysql_connect()` is deprecated, you should be using `mysql_real_connect()` anyway.)

To make `mysql_real_connect()` thread-safe, you must recompile the client library with this command:

```
shell> ./configure --enable-thread-safe-client
```

This creates a thread-safe client library `libmysqlclient_r`. (Assuming that your OS has a thread-safe `gethostbyname_r()` function.) This library is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Two threads can't send a query to the MySQL server at the same time on the same connection. In particular, you have to ensure that between a `mysql_query()` and `mysql_store_result()` no other thread is using the same connection.
- Many threads can access different result sets that are retrieved with `mysql_store_result()`.

- If you use `mysql_use_result`, you have to ensure that no other thread is using the same connection until the result set is closed. However, it really is best for threaded clients that share the same connection to use `mysql_store_result()`.
- If you want to use multiple threads on the same connection, you must have a mutex lock around your `mysql_query()` and `mysql_store_result()` call combination. Once `mysql_store_result()` is ready, the lock can be released and other threads may query the same connection.
- If you program with POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

You need to know the following if you have a thread that is calling MySQL functions which did not create the connection to the MySQL database:

When you call `mysql_init()` or `mysql_connect()`, MySQL creates a thread-specific variable for the thread that is used by the debug library (among other things).

If you call a MySQL function, before the thread has called `mysql_init()` or `mysql_connect()`, the thread does not have the necessary thread-specific variables in place and you are likely to end up with a core dump sooner or later.

To get things to work smoothly you have to do the following:

1. Call `my_init()` at the start of your program if it calls any other MySQL function before calling `mysql_real_connect()`.
2. Call `mysql_thread_init()` in the thread handler before calling any MySQL function.
3. In the thread, call `mysql_thread_end()` before calling `pthread_exit()`. This frees the memory used by MySQL thread-specific variables.

You may get some errors because of undefined symbols when linking your client with `libmysqlclient_r`. In most cases this is because you haven't included the thread libraries on the link/compile line.

22.3. MySQL PHP API

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic Web pages. It is available for most operating systems and Web servers, and can access most common databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with the Apache Web server.

PHP actually provides two different MySQL API extensions:

- `mysql`: Available for PHP versions 4 and 5, this extension is intended for use with MySQL versions prior to MySQL 4.1. This extension does not support the improved authentication protocol used in MySQL 5.0, nor does it support prepared statements or multiple statements. If you wish to use this extension with MySQL 5.0, you will likely want to configure the MySQL server to use the `--old-passwords` option (see [Section A.2.3, “Client does not support authentication protocol”](#)). This extension is documented on the PHP Web site at <http://php.net/mysql>.
- `mysqli` - Stands for “MySQL, Improved”; this extension is available only in PHP 5. It is intended for use with MySQL 4.1.1 and later. This extension fully supports the authentication protocol used in MySQL 5.0, as well as the Prepared Statements and Multiple Statements APIs. In addition, this extension provides an advanced, object-oriented programming interface. You can read the documentation for the `mysqli` extension at <http://php.net/mysqli>. A helpful article can be found at <http://www.zend.com/php5/articles/php5-mysqli.php>.

If you're experiencing problems with enabling both the `mysql` and the `mysqli` extension when building PHP on Linux yourself, see [Section 22.3.2, “Enabling Both `mysql` and `mysqli` in PHP”](#).

The PHP distribution and documentation are available from the [PHP Web site](#). MySQL provides the `mysql` and `mysqli` extensions for the Windows operating system for MySQL versions as of 5.0.18 on <http://dev.mysql.com/downloads/connector/php/>. You can find information why you should preferably use the extensions provided by MySQL on that page.

22.3.1. Common Problems with MySQL and PHP

- **Error: Maximum Execution Time Exceeded:** This is a PHP limit; go into the `php.ini` file and set the maximum execution time up from 30 seconds to something higher, as needed. It is also not a bad idea to double the RAM allowed per script to 16MB instead of 8MB.
- **Fatal error: Call to unsupported or undefined function `mysql_connect()` in ...:** This means that your PHP version isn't compiled with MySQL support. You can either compile a dynamic MySQL module and load it into PHP or recompile PHP with built-in MySQL support. This process is described in detail in the PHP manual.
- **Error: Undefined reference to 'uncompress':** This means that the client library is compiled with support for a compressed client/server protocol. The fix is to add `-lz` last when linking with `-lmysqlclient`.
- **Error: Client does not support authentication protocol:** This is most often encountered when trying to use the older `mysql` extension with MySQL 4.1.1 and later. Possible solutions are: downgrade to MySQL 4.0; switch to PHP 5 and the newer `mysqli` extension; or configure the MySQL server with `--old-passwords`. (See [Section A.2.3, "Client does not support authentication protocol"](#), for more information.)

Those with PHP4 legacy code can make use of a compatibility layer for the old and new MySQL libraries, such as this one:

<http://www.coggeshall.org/oss/mysql2i>.

22.3.2. Enabling Both `mysql` and `mysqli` in PHP

If you're experiencing problems with enabling both the `mysql` and the `mysqli` extension when building PHP on Linux yourself, you should try the following procedure.

1. Configure PHP like this:

```
./configure --with-mysqli=/usr/bin/mysql_config --with-mysql=/us
```

2. Edit the `Makefile` and search for a line that starts with `EXTRA_LIBS`. It might

look like this (all on one line):

```
EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl  
-lxml2 -lz -lm -lxml2 -lz -lm -lmysqlclient -lz -lcrypt -lnsl -l  
-lxml2 -lz -lm -lcrypt -lxml2 -lz -lm -lcrypt
```

Remove all duplicates, so that the line looks like this (all on one line):

```
EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl  
-lxml2
```

3. Build and install PHP:

```
make  
make install
```

22.4. MySQL Perl API

The Perl DBI module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI, you must install the DBI module, as well as a DataBase Driver (DBD) module for each type of server you want to access. For MySQL, this driver is the `DBD::mysql` module.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

Installation instructions for Perl DBI support are given in [Section 2.14, “Perl Installation Notes”](#).

DBI information is available at the command line, online, or in printed form:

- Once you have the DBI and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

```
shell> perldoc DBI
shell> perldoc DBI::FAQ
shell> perldoc DBD::mysql
```

You can also use `pod2man`, `pod2html`, and so forth to translate this information into other formats.

- For online information about Perl DBI, visit the DBI Web site, <http://dbi.perl.org/>. That site hosts a general DBI mailing list. MySQL AB hosts a list specifically about `DBD::mysql`; see [Section 1.7.1, “MySQL Mailing Lists”](#).
- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI Web site, <http://dbi.perl.org/>.

For information that focuses specifically on using DBI with MySQL, see *MySQL and Perl for the Web* (Paul DuBois, New Riders, 2001). This book's Web site is <http://www.kitebird.com/mysql-perl/>.

22.5. MySQL C++ API

MySQL++ is a MySQL API for C++. Warren Young has taken over this project. More information can be found at <http://www.mysql.com/products/mysql++/>.

22.5.1. Borland C++

You can compile the MySQL Windows source with Borland C++ 5.02. (The Windows source includes only projects for Microsoft VC++, for Borland C++ you have to do the project files yourself.)

One known problem with Borland C++ is that it uses a different structure alignment than VC++. This means that you run into problems if you try to use the default `libmysql.dll` libraries (that were compiled using VC++) with Borland C++. To avoid this problem, only call `mysql_init()` with `NULL` as an argument, not a pre-allocated `MYSQL` structure.

22.6. MySQL Python API

MySQLdb provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at <http://sourceforge.net/projects/mysql-python/>.

22.7. MySQL Tcl API

MySQLtcl is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at <http://www.xdobry.de/mysqltcl/>.

22.8. MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at <http://efsa.sourceforge.net/archive/ravits/mysql.htm>.

22.9. MySQL Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

- `msql2mysql`

A shell script that converts mSQL programs to MySQL. It doesn't handle every case, but it gives a good start when converting.

- `mysql_config`

A shell script that produces the option values needed when compiling MySQL programs.

22.9.1. `msql2mysql` — Convert mSQL Programs for Use with MySQL

Initially, the MySQL C API was developed to be very similar to that for the mSQL database system. Because of this, mSQL programs often can be converted relatively easily for use with MySQL by changing the names of the C API functions.

The **`msql2mysql`** utility performs the conversion of mSQL C API function calls to their MySQL equivalents. **`msql2mysql`** converts the input file in place, so make a copy of the original before converting it. For example, use **`msql2mysql`** like this:

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

Then examine `client-prog.c` and make any post-conversion revisions that may be necessary.

`msql2mysql` uses the **`replace`** utility to make the function name substitutions. See [Section 8.18, “`replace` — A String-Replacement Utility”](#).

22.9.2. `mysql_config` — Get Compile Options for Compiling Clients

`mysql_config` provides you with useful information for compiling your MySQL client and connecting it to MySQL.

`mysql_config` supports the following options:

- `--cflags`

Compiler flags to find include files and critical compiler flags and defines used when compiling the `libmysqlclient` library.

- `--include`

Compiler options to find MySQL include files. (Note that normally you would use `--cflags` instead of this option.)

- `--libmysqld-libs`, `--embedded`

Libraries and options required to link with the MySQL embedded server.

- `--libs`

Libraries and options required to link with the MySQL client library.

- `--libs_r`

Libraries and options required to link with the thread-safe MySQL client library.

- `--port`

The default TCP/IP port number, defined when configuring MySQL.

- `--socket`

The default Unix socket file, defined when configuring MySQL.

- `--version`

Version number for the MySQL distribution.

If you invoke **mysql_config** with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
  --cflags          [-I/usr/local/mysql/include/mysql -mcpu=pentiumpr
  --include         [-I/usr/local/mysql/include/mysql]
  --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz
                  -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                  -lpthread -lz -lcrypt -lnsl -lm -lpthread]
  --socket         [/tmp/mysql.sock]
  --port           [3306]
  --version        [4.0.16]
  --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread
                  -lcrypt -lnsl -lm -lpthread -lrt]
```

You can use **mysql_config** within a command line to include the value that it displays for a particular option. For example, to compile a MySQL client program, use **mysql_config** as follows:

```
shell> CFG=/usr/local/mysql/bin/mysql_config
shell> sh -c "gcc -o progname ` $CFG --cflags ` progname.c ` $CFG --lib
```

When you use **mysql_config** this way, be sure to invoke it within backtick (``) characters. That tells the shell to execute it and substitute its output into the surrounding command.

Chapter 23. Connectors

Table of Contents

- [23.1. MySQL Connector/ODBC](#)
 - [23.1.1. Introduction to MyODBC](#)
 - [23.1.2. How to Install MyODBC](#)
 - [23.1.3. MyODBC Configuration](#)
 - [23.1.4. MyODBC Examples](#)
 - [23.1.5. MyODBC Reference](#)
 - [23.1.6. MyODBC Notes and Tips](#)
 - [23.1.7. MyODBC Support](#)
- [23.2. Connector/NET](#)
 - [23.2.1. Connector/NET Versions](#)
 - [23.2.2. How to install Connector/NET](#)
 - [23.2.3. Connector/NET Examples](#)
 - [23.2.4. Connector/NET Reference](#)
 - [23.2.5. Connector/NET Notes and Tips](#)
 - [23.2.6. Connector/NET Support](#)
- [23.3. MySQL Connector/J](#)
 - [23.3.1. Connector/J Versions](#)
 - [23.3.2. Installing Connector/J](#)
 - [23.3.3. Connector/J Examples](#)
 - [23.3.4. Connector/J \(JDBC\) Reference](#)
 - [23.3.5. Connector/J Notes and Tips](#)
 - [23.3.6. Connector/J Support](#)
- [23.4. MySQL Connector/MXJ](#)
 - [23.4.1. Introduction to Connector/MXJ](#)
 - [23.4.2. Installing Connector/MXJ](#)
 - [23.4.3. Connector/MXJ Configuration](#)
 - [23.4.4. Connector/MXJ Reference](#)
 - [23.4.5. Connector/MXJ Notes and Tips](#)
 - [23.4.6. Connector/MXJ Support](#)
- [23.5. Connector/PHP](#)

This chapter describes MySQL Connectors, drivers that provide connectivity to the MySQL server for client programs. There are currently five MySQL

Connectors:

- Connector/ODBC provides driver support for connecting to a MySQL server using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix and Mac OS X platforms.
- Connector/NET enables developers to create .NET applications that use data stored in a MySQL database. Connector/NET implement a fully-functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that want to use Connector/NET can be written in any of the supported .NET languages.
- Connector/J provides driver support for connecting to MySQL from a Java application using the standard Java Database Connectivity (JDBC) API.
- Connector/MXJ is a tool that enables easy deployment and management of MySQL server and database through your Java application.
- Connector/PHP is a Windows-only connector for PHP that provides the `mysql` and `mysql_i` extensions for use with MySQL 5.0.18 and later.

For information on connecting to a MySQL server using other languages and interfaces than those detailed above, including Perl, Python and PHP for other platforms and environments, please refer to the [Chapter 22, APIs and Libraries](#) chapter.

23.1. MySQL Connector/ODBC

The MySQL Connector/ODBC is the name for the family of MySQL ODBC drivers (also called MyODBC drivers) that provide access to a MySQL database using the industry standard Open Database Connectivity (ODBC) API. This reference covers Connector/ODBC 3.51, a version of the API that provides ODBC 3.5x compliant access to a MySQL database.

The manual for versions of MyODBC older than 3.51 can be located in the corresponding binary or source distribution.

For more information on the ODBC API standard and how to use it, refer to <http://www.microsoft.com/data/>.

The application development part of this reference assumes a good working knowledge of C, general DBMS knowledge, and finally, but not least, familiarity with MySQL. For more information about MySQL functionality and its syntax, refer to <http://dev.mysql.com/doc/>.

Typically, you need to install MyODBC only on Windows machines. For Unix and Mac OS X you can use the native MySQL network or named pipe to communicate with your MySQL database. You may need MyODBC for Unix or Mac OS X if you have an application that requires an ODBC interface to communicate with database.. Applications that require ODBC to communicate with MySQL include ColdFusion, Microsoft Office, and Filemaker Pro.

If you want to install the MyODBC connector on a Unix host, then you must also install an ODBC manager.

If you have questions that are not answered in this document, please send a mail message to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com).

23.1.1. Introduction to MyODBC

ODBC (Open Database Connectivity) provides a way for client programs to access a wide range of databases or data sources. ODBC is a standardized API that allows connections to SQL database servers. It was developed according to the specifications of the SQL Access Group and defines a set of function calls,

error codes, and data types that can be used to develop database-independent applications. ODBC usually is used when database independence or simultaneous access to different data sources is required.

For more information about ODBC, refer to <http://www.microsoft.com/data/>.

23.1.1.1. MyODBC Versions

There are currently two version of MyODBC available:

- MyODBC 5.0, currently in beta status, has been designed to extend the functionality of the MyODBC 3.51 driver and incorporate full support for the functionality in the MySQL 5.0 server release, including stored procedures and views. Applications using MyODBC 3.51 will be compatible with MyODBC 5.0, while being able to take advantage of the new features. Features and functionality of the MyODBC 5.0 driver are not currently included in this guide.
- MyODBC 3.51 is the current release of the 32-bit ODBC driver, also known as the MySQL ODBC 3.51 driver. This version is enhanced compared to the older MyODBC 2.50 driver. It has support for ODBC 3.5x specification level 1 (complete core API + level 2 features) in order to continue to provide all functionality of ODBC for accessing MySQL.
- MyODBC 2.50 is the previous version of the 32-bit ODBC driver from MySQL AB that is based on ODBC 2.50 specification level 0 (with level 1 and 2 features). Information about the MyODBC 2.50 driver is included in this guide for the purposes of comparison only.

Note: From this section onward, the primary focus of this guide is the MyODBC 3.51 driver. More information about the MyODBC 2.50 driver in the documentation included in the installation packages for that version. If there is a specific issue (error or known problem) that only affects the 2.50 version, it may be included here for reference.

23.1.1.2. General Information About ODBC and MyODBC

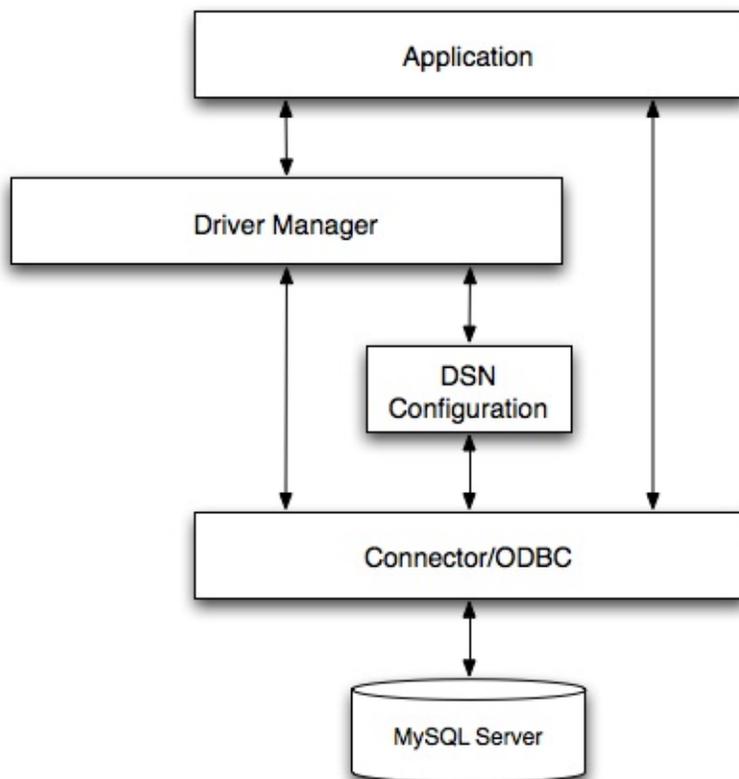
Open Database Connectivity (ODBC) is a widely accepted application-programming interface (API) for database access. It is based on the Call-Level

Interface (CLI) specifications from X/Open and ISO/IEC for database APIs and uses Structured Query Language (SQL) as its database access language.

A survey of ODBC functions supported by MyODBC is given at [Section 23.1.5.1, “MyODBC API Reference”](#). For general information about ODBC, see <http://www.microsoft.com/data/>.

23.1.1.2.1. MyODBC Architecture

The MyODBC architecture is based on five components, as shown in the following diagram:



- **Application:**

The Application uses the ODBC API to access the data from the MySQL server. The ODBC API in turn communicates with the Driver Manager. The Application communicates with the Driver Manager using the standard ODBC calls. The Application does not care where the data is stored, how it is stored, or even how the system is configured to access the

data. It needs to know only the Data Source Name (DSN).

A number of tasks are common to all applications, no matter how they use ODBC. These tasks are:

- Selecting the MySQL server and connecting to it
- Submitting SQL statements for execution
- Retrieving results (if any)
- Processing errors
- Committing or rolling back the transaction enclosing the SQL statement
- Disconnecting from the MySQL server

Because most data access work is done with SQL, the primary tasks for applications that use ODBC are submitting SQL statements and retrieving any results generated by those statements.

- **Driver manager:**

The Driver Manager is a library that manages communication between application and driver or drivers. It performs the following tasks:

- Resolves Data Source Names (DSN). The DSN is a configuration string that identifies a given database driver, database, database host and optionally authentication information that enables an ODBC application to connect to a database using a standardized reference.

Because the database connectivity information is identified by the DSN, any ODBC compliant application can connect to the data source using the same DSN reference. This eliminates the need to separately configure each application that needs access to a given database; instead you instruct the application to use a pre-configured DSN.

- Loading and unloading of the driver required to access a specific database as defined within the DSN. For example, if you have

configured a DSN that connects to a MySQL database then the driver manager will load the MyODBC driver to enable the ODBC API to communicate with the MySQL host.

- Processes ODBC function calls or passes them to the driver for processing.

- **MyODBC Driver:**

The MyODBC driver is a library that implements the functions supported by the ODBC API. It processes ODBC function calls, submits SQL requests to MySQL server, and returns results back to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by MySQL.

- **DSN Configuration:**

The ODBC configuration file stores the driver and database information required to connect to the server. It is used by the Driver Manager to determine which driver to be loaded according to the definition in the DSN. The driver uses this to read connection parameters based on the DSN specified. For more information, [Section 23.1.3, “MyODBC Configuration”](#).

- **MySQL Server:**

The MySQL database where the information is stored. The database is used as the source of the data (during queries) and the destination for data (during inserts and updates).

23.1.1.2.2. ODBC Driver Managers

An ODBC Driver Manager is a library that manages communication between the ODBC-aware application and any drivers. Its main functionality includes:

- Resolving Data Source Names (DSN).
- Driver loading and unloading.
- Processing ODBC function calls or passing them to the driver.

Both Windows and Mac OS X include ODBC driver managers with the operating system. Most ODBC Driver Manager implementations also include an administration application that makes the configuration of DSN and drivers easier. Examples and information on these managers, including Unix ODBC driver managers are listed below:

- Microsoft Windows ODBC Driver Manager (odbc32.dll), <http://www.microsoft.com/data/>.
- Mac OS X includes ODBC Administrator, a GUI application that provides a simpler configuration mechanism for the Unix iODBC Driver Manager. You can configure DSN and driver information either through ODBC Administrator or through the iODBC configuration files. This also means that you can test ODBC Administrator configurations using the `iodbctest` command. <http://www.apple.com>.
- unixODBC Driver Manager for Unix (`libodbc.so`). See <http://www.unixodbc.org>, for more information. The unixODBC Driver Manager includes the MyODBC driver 3.51 in the installation package, starting with version unixODBC 2.1.2.
- iODBC ODBC Driver Manager for Unix (`libiodbc.so`), see <http://www.iodbc.org>, for more information.

23.1.2. How to Install MyODBC

You can install the MyODBC drivers using two different methods, a binary installation and a source installation. The binary installation is the easiest and most straightforward method of installation. Using the source installation methods should only be necessary on platforms where a binary installation package is not available, or in situations where you want to customize or modify the installation process or MyODBC drivers before installation.

23.1.2.1. Where to Get MyODBC

MySQL AB distributes all its products under the General Public License (GPL). You can get a copy of the latest version of MyODBC binaries and sources from the MySQL AB Web site <http://dev.mysql.com/downloads/>.

For more information about MyODBC, visit <http://www.mysql.com/products/myodbc/>.

For more information about licensing, visit <http://www.mysql.com/company/legal/licensing/>.

23.1.2.2. Supported Platforms

MyODBC can be used on all major platforms supported by MySQL. You can install it on:

- Windows 95, 98, Me, NT, 2000, XP, and 2003
- All Unix-like Operating Systems, including: AIX, Amiga, BSDI, DEC, FreeBSD, HP-UX 10/11, Linux, NetBSD, OpenBSD, OS/2, SGI Irix, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64 Unix
- Mac OS X and Mac OS X Server

If a binary distribution is not available for a particular platform, see [Section 23.1.2.4, “Installing MyODBC from a source distribution”](#), to build the driver from the original source code. You can contribute the binaries you create to MySQL by sending a mail message to <myodbc@lists.mysql.com>, so that it becomes available for other users.

23.1.2.3. Installing MyODBC from a binary distribution

Using a binary distribution offers the most straightforward method for installing MyODBC. If you want more control over the driver, the installation location and or to customize elements of the driver you will need to build and install from the source. See the [Section 23.1.2.4, “Installing MyODBC from a source distribution”](#).

23.1.2.3.1. Installing MyODBC from a Binary Distribution on Windows

Before installing the MyODBC drivers on Windows you should ensure that your Microsoft Data Access Components (MDAC) are up to date. You can obtain the latest version from the [Microsoft Data Access and Storage](#) website.

There are three available distribution types to use when installing for Windows. The contents in each case are identical, it is only the installation method which is different.

- Zipped installer consists of a Zipped package containing a standalone installation application. To install from this package, you must unzip the installer, and then run the installation application. See [Section 23.1.2.3.1.1, “Installing the Windows MyODBC Driver using an installer”](#) to complete the installation.
- MSI installer, an installation file that can be used with the installer included in Windows 2000, Windows XP and Windows Server 2003. See [Section 23.1.2.3.1.1, “Installing the Windows MyODBC Driver using an installer”](#) to complete the installation.
- Zipped DLL package, containing the DLL files that need must be manually installed. See [Section 23.1.2.3.1.2, “Installing the Windows MyODBC Driver using the Zipped DLL package”](#) to complete the installation.

23.1.2.3.1.1. Installing the Windows MyODBC Driver using an installer

The installer packages offer a very simple method for installing the MyODBC drivers. If you have downloaded the zipped installer then you must extract the installer application. The basic installation process is identical for both installers.

You should follow these steps to complete the installation:

1. Double click on the standalone installer that you extracted, or the MSI file you downloaded.
2. The MySQL Connector/ODBC 3.51 - Setup Wizard will start. Click the Next button to begin the installation process.

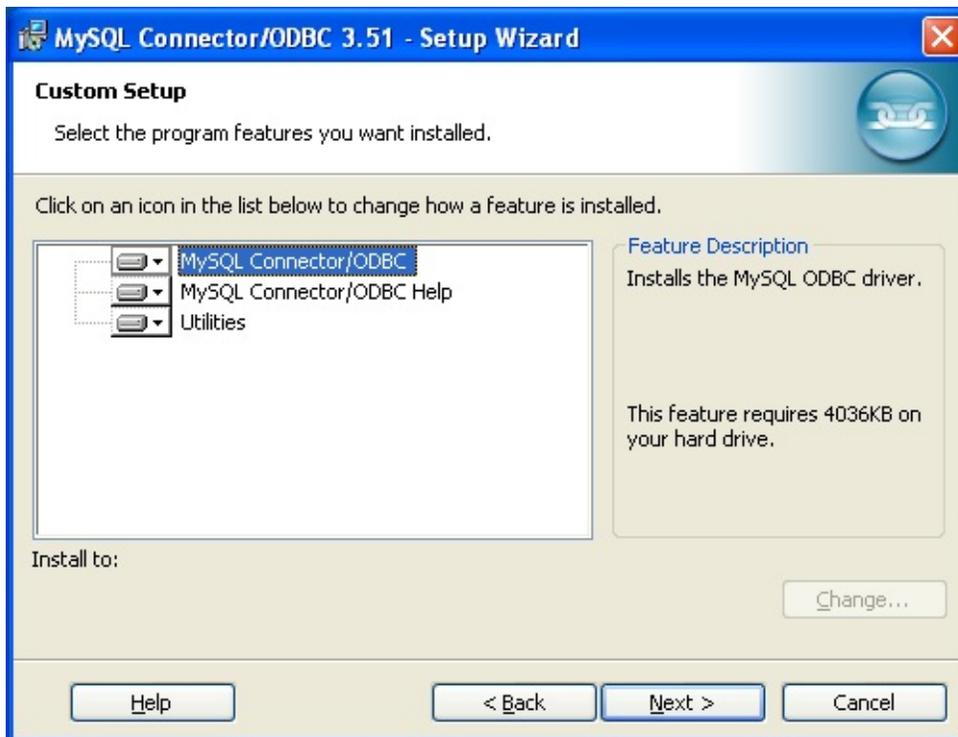


3. You will need to choose the installation type. The Typical installation provides the standard files you will need to connect to a MySQL database using ODBC. The Complete option installs all the available files, including debug and utility components. It is recommended you choose one of these two options to complete the installation. If choose one of these methods, click Next and then proceed to step 5.

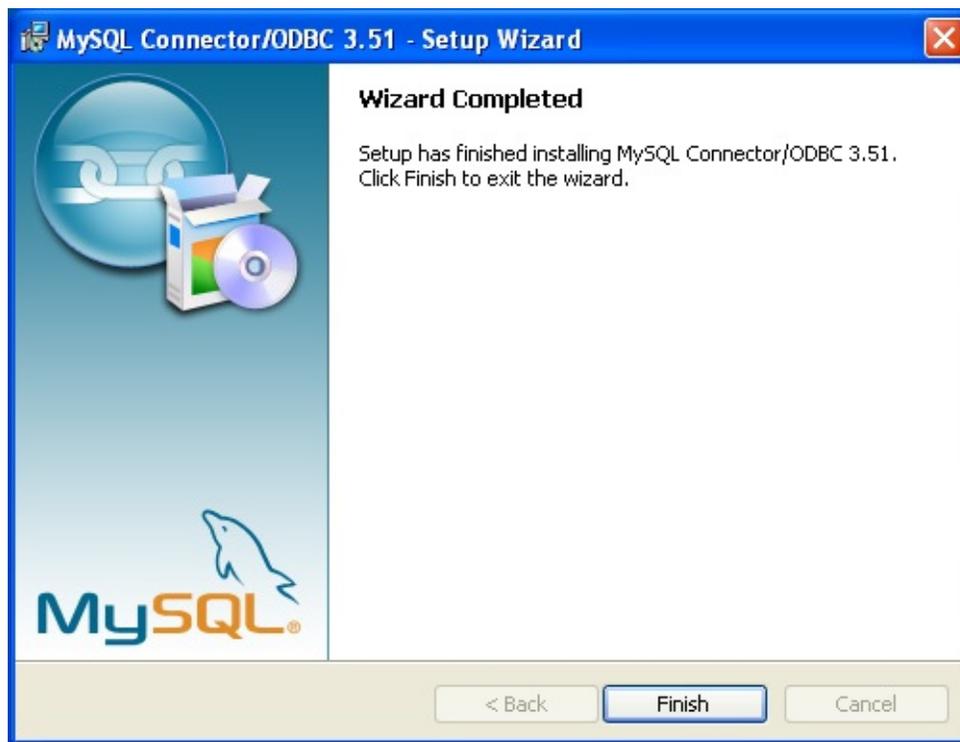
You may also choose a Custom installation, which enables you to select the individual components that you want to install. You have chosen this method, click Next and then proceed to step 4.



4. If you have chosen a custom installation, use the popups to select which components to install and then click Next to install the necessary files.



5. Once the files have copied to your machine, the installation is complete. Click Finish to exit the installer.



Now the installation is complete, you can continue to configure your ODBC connections using [Section 23.1.3, “MyODBC Configuration”](#).

23.1.2.3.1.2. Installing the Windows MyODBC Driver using the Zipped DLL package

If you have downloaded the Zipped DLL package then you must install the individual files required for MyODBC operation manually. Once you have unzipped the installation files, you can either perform this operation by hand, executing each statement individually, or you can use the included Batch file to perform an installation to the default locations.

To install using the Batch file:

1. Unzip the MyODBC Zipped DLL package.
2. Open a Command Prompt.
3. Change to the directory created when you unzipped the MyODBC Zipped DLL package.

4. Run `Install.bat`:

```
C:\> Install.bat
```

This will copy the necessary files into the default location, and then register the MyODBC driver with the Windows ODBC manager.

If you want to copy the files to an alternative location - for example, to run or test different versions of the MyODBC driver on the same machine, then you must copy the files by hand. It is however not recommended to install these files in a non-standard location. To copy the files by hand to the default installation location use the following steps:

1. Unzip the MyODBC Zipped DLL package.
2. Open a Command Prompt.
3. Change to the directory created when you unzipped the MyODBC Zipped DLL package.
4. Copy the library files to a suitable directory. The default is to copy them into the default Windows system directory `\Windows\System32`:

```
C:\> copy lib\myodbc3S.dll \Windows\System32  
C:\> copy lib\myodbc3S.lib \Windows\System32  
C:\> copy lib\myodbc3.dll \Windows\System32  
C:\> copy lib\myodbc3.lib \Windows\System32
```

5. Copy the MyODBC tools. These must be placed into a directory that is in the system `PATH`. The default is to install these into the Windows system directory `\Windows\System32`:

```
C:\> copy bin\myodbc3i.exe \Windows\System32  
C:\> copy bin\myodbc3m.exe \Windows\System32  
C:\> copy bin\myodbc3c.exe \Windows\System32
```

6. Optionally copy the help files. For these files to be accessible through the help system, they must be installed in the Windows system directory:

```
C:\> copy doc\*.hlp \Windows\System32
```

7. Finally, you must register the MyODBC driver with the ODBC manager:

```
C:\> myodbc3i -a -d -t"MySQL ODBC 3.51 Driver;\
DRIVER=myodbc3.dll;SETUP=myodbc3S.dll"
```

You must change the references to the DLL files and command location in the above statement if you have not installed these files into the default location.

23.1.2.3.1.3. Handling Installation Errors

On Windows, you may get the following error when trying to install the older MyODBC 2.50 driver:

```
An error occurred while copying C:\WINDOWS\SYSTEM\MFC30.DLL.
Restart Windows and try installing again (before running any
applications which use ODBC)
```

The reason for the error is that another application is currently using the ODBC system. Windows may not allow you to complete the installation. In most cases, you can continue by pressing Ignore to copy the rest of the MyODBC files and the final installation should still work. If it doesn't, the solution is to re-boot your computer in "safe mode." Choose safe mode by pressing F8 just before your machine starts Windows during re-booting, install the MyODBC drivers, and re-boot to normal mode.

23.1.2.3.2. Installing MyODBC from a Binary Distribution on Unix

There are two methods available for installing MyODBC on Unix from a binary distribution. For most Unix environments you will need to use the tarball distribution. For Linux systems, there is also an RPM distribution available.

23.1.2.3.2.1. Installing MyODBC from a Binary Tarball Distribution

To install the driver from a tarball distribution (.tar.gz file), download the latest version of the driver for your operating system and follow these steps that demonstrate the process using the Linux version of the tarball:

```
shell> su root
shell> gunzip MyODBC-3.51.11-i686-pc-linux.tar.gz
shell> tar xvf MyODBC-3.51.11-i686-pc-linux.tar
shell> cd MyODBC-3.51.11-i686-pc-linux
```

Read the installation instructions in the INSTALL - BINARY file and execute these commands.

```
shell> cp libmyodbc* /usr/local/lib
shell> cp odbc.ini /usr/local/etc
shell> export ODBCINI=/usr/local/etc/odbc.ini
```

Then proceed on to [Section 23.1.3.4, “Configuring a MyODBC DSN on Unix”](#), to configure the DSN for MyODBC. For more information, refer to the INSTALL - BINARY file that comes with your distribution.

23.1.2.3.2.2. Installing MyODBC from an RPM Distribution

To install or upgrade MyODBC from an RPM distribution on Linux, simply download the RPM distribution of the latest version of MyODBC and follow the instructions below. Use **su root** to become root, then install the RPM file.

If you are installing for the first time:

```
shell> su root
shell> rpm -ivh MyODBC-3.51.12.i386.rpm
```

If the driver exists, upgrade it like this:

```
shell> su root
shell> rpm -Uvh MyODBC-3.51.12.i386.rpm
```

If there is any dependency error for MySQL client library, libmysqlclient, simply ignore it by supplying the `--nodeps` option, and then make sure the MySQL client shared library is in the path or set through `LD_LIBRARY_PATH`.

This installs the driver libraries and related documents to `/usr/local/lib` and `/usr/share/doc/MyODBC`, respectively. Proceed onto [Section 23.1.3.4, “Configuring a MyODBC DSN on Unix”](#).

To **uninstall** the driver, become root and execute an **rpm** command:

```
shell> su root
shell> rpm -e MyODBC
```

23.1.2.3.3. Installing MyODBC on Mac OS X

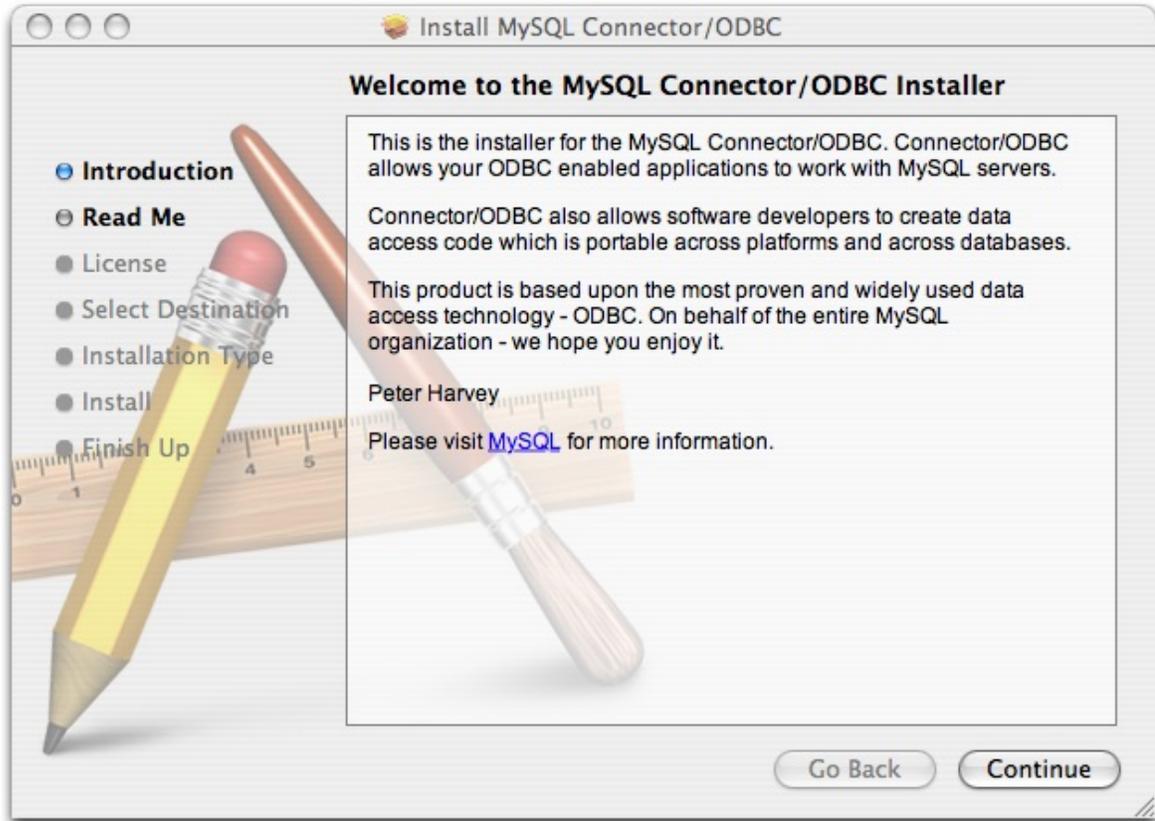
Mac OS X is based on the FreeBSD operating system, and you can normally use the MySQL network port for connecting to MySQL servers on other hosts. Installing the MyODBC driver enables you to connect to MySQL databases on any platform through the ODBC interface. You should only need to install the MyODBC driver when your application requires an ODBC interface. Applications that require or can use ODBC (and therefore the MyODBC driver) include ColdFusion, Filemaker Pro, 4th Dimension and many other applications.

Mac OS X includes its own ODBC manager, based on the iODBC manager. Mac OS X includes an administration tool that provides easier administration of ODBC drivers and configuration, updating the underlying iODBC configuration files.

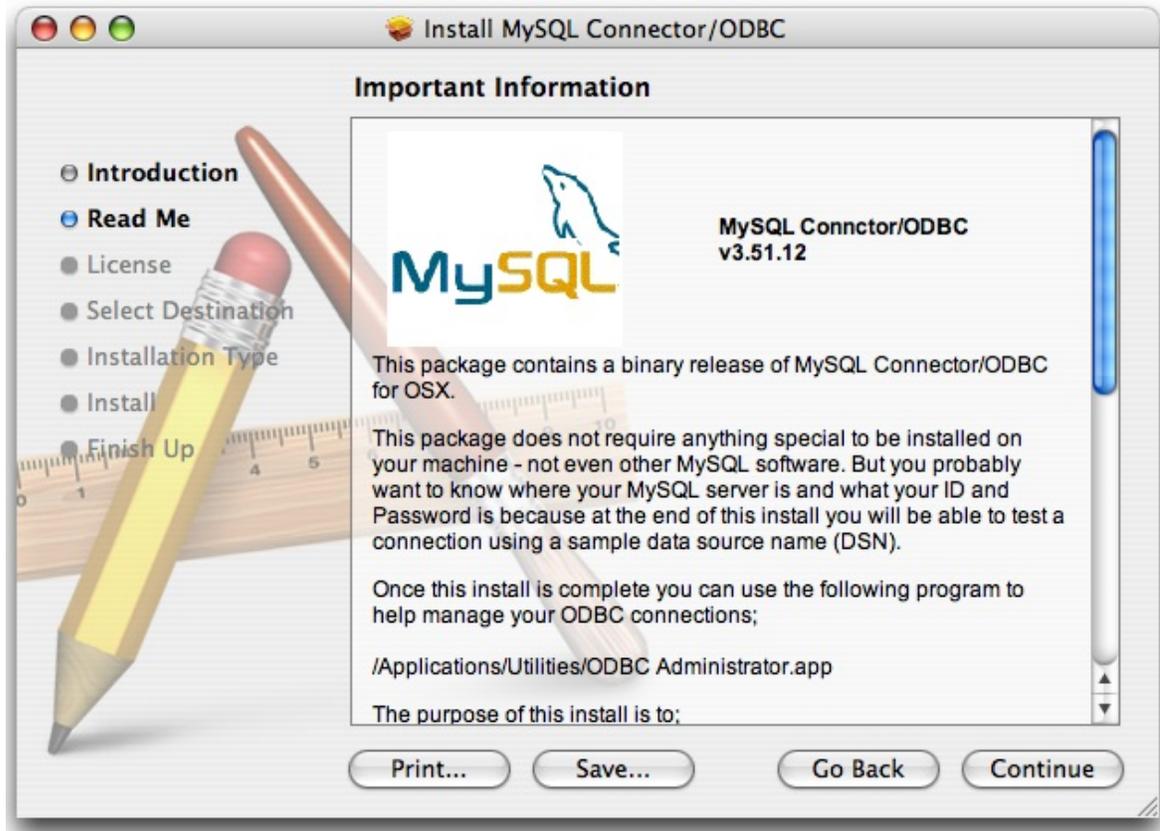
23.1.2.3.3.1. Installing the MyODBC Driver

You can install MyODBC on a Mac OS X or Mac OS X Server computer by using the binary distribution. The package is available as a compressed disk image (.dmg) file. To install MyODBC on your computer using this method, follow these steps:

1. Download the file to your computer and double-click on the downloaded image file.
2. Within the disk image you will find an installer package (with the .pkg extension). Double click on this file to start the Mac OS X installer.
3. You will be presented with the installer welcome message. Click the Continue button to begin the installation process.



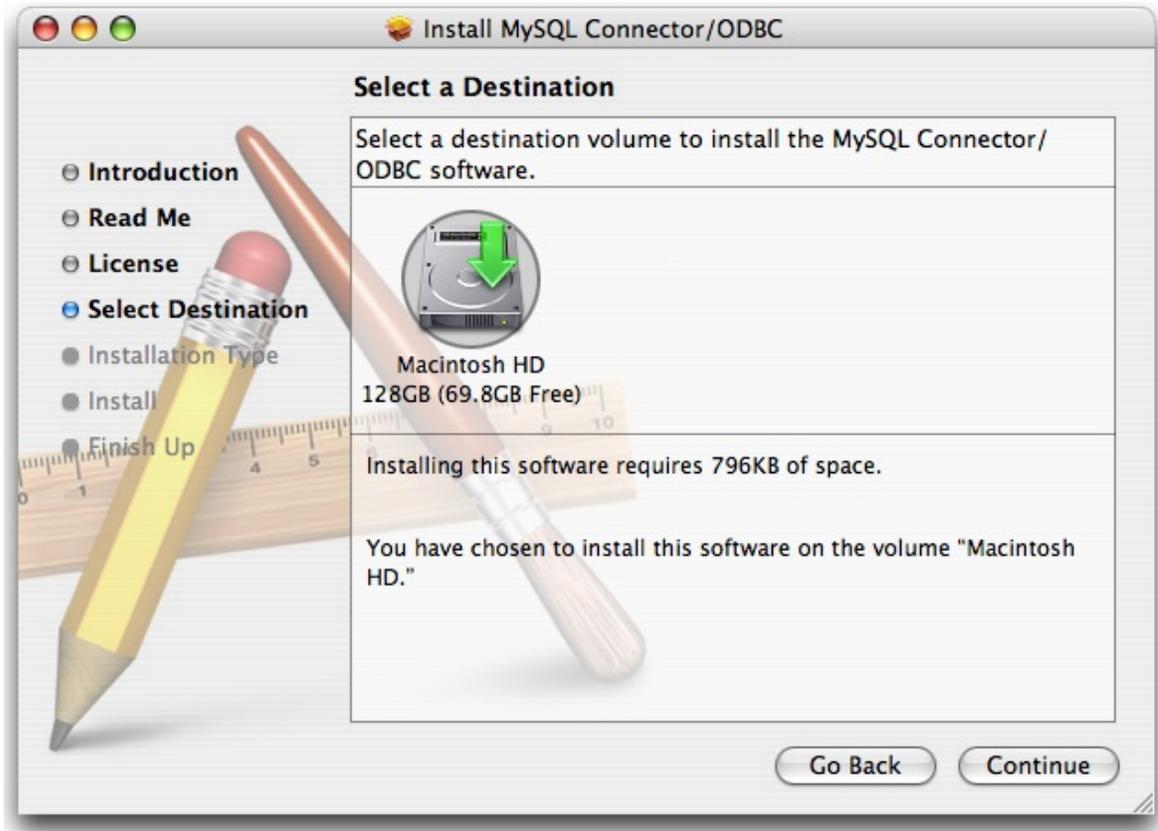
4. Please take the time to read the Important Information as it contains guidance on how to complete the installation process. Once you have read the notice and collected the necessary information, click Continue.



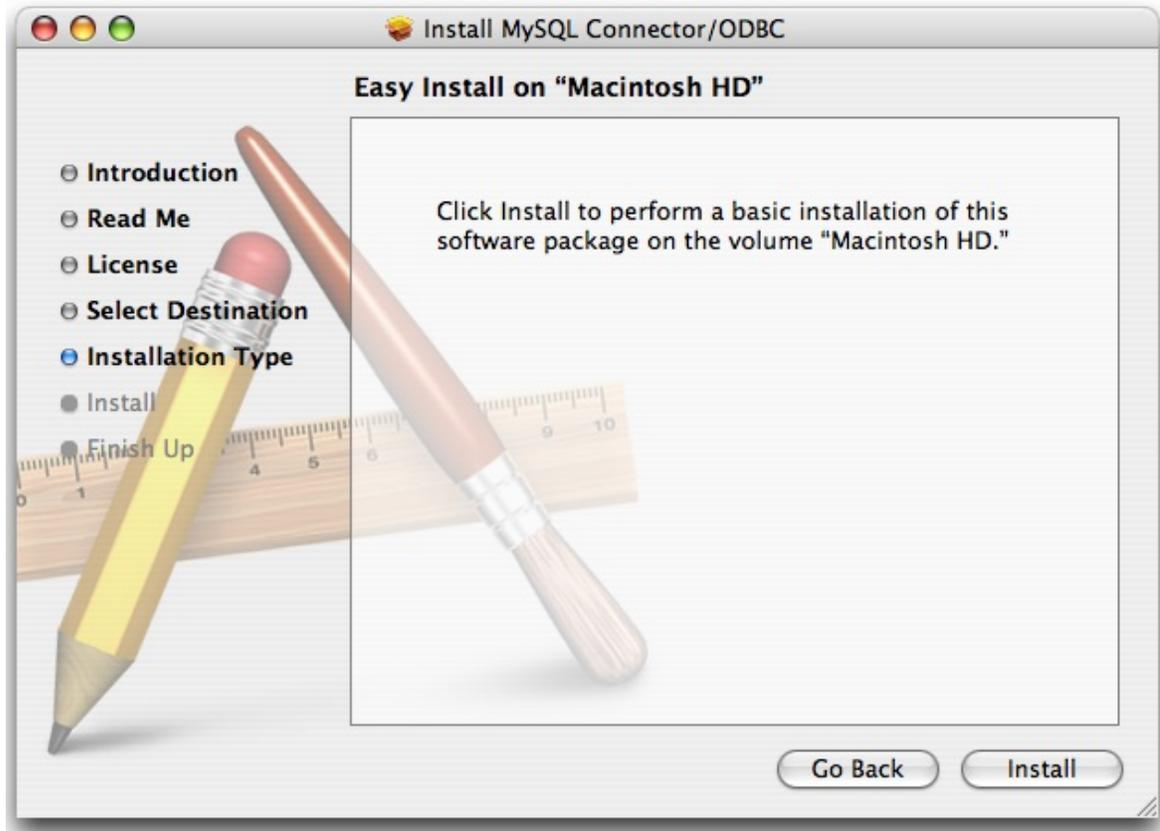
5. MyODBC drivers are made available under the GNU General Public License. Please read the license if you are not familiar with it before continuing installation. Click Continue to approve the license (you will be asked to confirm that decision) and continue the installation.



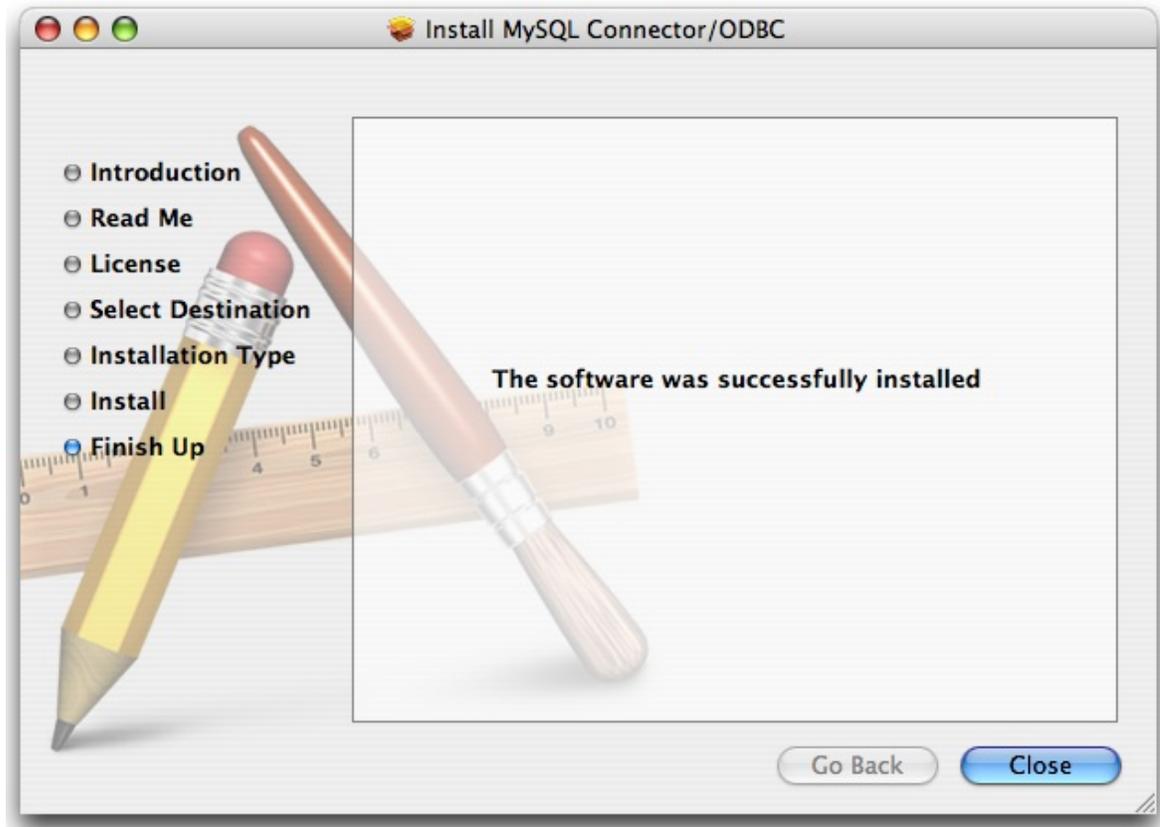
6. Choose a location to install the MyODBC drivers and the ODBC Administrator application. You must install the files onto a drive with an operating system and you may be limited in the choices available. Select the drive you want to use, and then click Continue.



7. The installer will automatically select the files that need to be installed on your machine. Click Install to continue. The installer will copy the necessary files to your machine. A progress bar will be shown indicating the installation progress.



8. When installation has been completed you will get a window like the one shown below. Click Close to close and quit the installer.



23.1.2.4. Installing MyODBC from a source distribution

Installing MyODBC from a source distribution gives you greater flexibility in the contents and installation location of the MyODBC components. It also enables you to build and install MyODBC on platforms where a pre-compiled binary is not available.

MyODBC sources are available either as a downloadable package, or through the revision control system used by the MyODBC developers.

23.1.2.4.1. Installing MyODBC from a Source Distribution on Windows

You should only need to install MyODBC from source on Windows if you want to change or modify the source or installation. If you are unsure whether to install from source, please use the binary installation detailed in [Section 23.1.2.3.1, “Installing MyODBC from a Binary Distribution on Windows”](#).

Installing MyODBC from source on Windows requires a number of different

tools and packages:

- MDAC, Microsoft Data Access SDK from <http://www.microsoft.com/data/>.
- Suitable C compiler, such as Microsoft Visual C++ or the C compiler included with Microsoft Visual Studio.
- Compatible make tool. Microsoft's nmake is used in the examples in this section.
- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because MyODBC uses new calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit <http://dev.mysql.com/downloads/>.

23.1.2.4.1.1. Building MyODBC 3.51

MyODBC source distributions include `Makefile`s that require the **nmake** or other make utility. In the distribution, you can find `Makefile` for building the release version and `Makefile_debug` for building debugging versions of the driver libraries and DLLs.

To build the driver, use this procedure:

1. Download and extract the sources to a folder, then change directory into that folder. The following command assumes the folder is named `myodbc3-src`:

```
C:\> cd myodbc3-src
```

2. Edit `Makefile` to specify the correct path for the MySQL client libraries and header files. Then use the following commands to build and install the release version:

```
C:\> nmake -f Makefile
C:\> nmake -f Makefile install
```

nmake -f Makefile builds the release version of the driver and places the binaries in subdirectory called `Release`.

nmake -f Makefile install installs (copies) the driver DLLs and libraries (myodbc3.dll, myodbc3.lib) to your system directory.

3. To build the debug version, use `Makefile_Debug` rather than `Makefile`, as shown below:

```
C:\> nmake -f Makefile_debug  
C:\> nmake -f Makefile_debug install
```

4. You can clean and rebuild the driver by using:

```
C:\> nmake -f Makefile clean  
C:\> nmake -f Makefile install
```

Note:

- Make sure to specify the correct MySQL client libraries and header files path in the Makefiles (set the `MYSQL_LIB_PATH` and `MYSQL_INCLUDE_PATH` variables). The default header file path is assumed to be `C:\mysql\include`. The default library path is assumed to be `C:\mysql\lib\opt` for release DLLs and `C:\mysql\lib\debug` for debug versions.
- For the complete usage of **nmake**, visit http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vcce4/html/evgrfRunningNMAKE.asp.
- If you are using the Subversion tree for compiling, all Windows-specific Makefiles are named as `win_Makefile*`.

23.1.2.4.1.2. Testing

After the driver libraries are copied/installed to the system directory, you can test whether the libraries are properly built by using the samples provided in the `samples` subdirectory:

```
C:\> cd samples  
C:\> nmake -f Makefile all
```

23.1.2.4.1.3. Building MyODBC 2.50

The MyODBC 2.50 source distribution includes VC workspace files. You can

build the driver using these files (.dsp and .dsw) directly by loading them from Microsoft Visual Studio 6.0 or higher.

23.1.2.4.2. Installing MyODBC from a Source Distribution on Unix

You need the following tools to build MySQL from source on Unix:

- A working ANSI C++ compiler. **gcc** 2.95.2 or later, **egcs** 1.0.2 or later or **egcs 2.91.66**, SGI C++, and SunPro C++ are some of the compilers that are known to work.
- A good **make** program. GNU **make** is always recommended and is sometimes required.
- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because MyODBC uses new calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit <http://dev.mysql.com/downloads/>.

If you have built your own MySQL server and/or client libraries from source then you must have used the `--enable-thread-safe-client` option to configure when the libraries were built.

You should also ensure that the `libmysqlclient` library were built and installed as a shared library.

- A compatible ODBC manager must be installed. MyODBC is known to work with the `iODBC` and `unixODBC` managers. See [Section 23.1.1.2.2, “ODBC Driver Managers”](#), for more information.
- If you are using a character set that isn't compiled into the MySQL client library then you need to install the MySQL character definitions from the `charsets` directory into `SHAREDIR` (by default, `/usr/local/mysql/share/mysql/charsets`). These should be in place if you have installed the MySQL server on the same machine. See [Chapter 10, *Character Set Support*](#), for more information on character set support.

Once you have all the required files, unpack the source files to a separate directory, you then have to run **configure** and build the library using **make**.

23.1.2.4.2.1. Typical configure Options

The **configure** script gives you a great deal of control over how you configure your MyODBC build. Typically you do this using options on the **configure** command line. You can also affect **configure** using certain environment variables. For a list of options and environment variables supported by **configure**, run this command:

```
shell> ./configure --help
```

Some of the more commonly used **configure** options are described here:

1. To compile MyODBC, you need to supply the MySQL client include and library files path using the `--with-mysql-path=DIR` option, where *DIR* is the directory where MySQL is installed.

MySQL compile options can be determined by running `DIR/bin/mysql_config`.

2. Supply the standard header and library files path for your ODBC Driver Manager (iODBC or unixODBC).
 - If you are using iODBC and iODBC is not installed in its default location (`/usr/local`), you might have to use the `--with-iodbc=DIR` option, where *DIR* is the directory where iODBC is installed.

If the iODBC headers do not reside in `DIR/include`, you can use the `--with-iodbc-includes=INCDIR` option to specify their location.

The applies to libraries. If they are not in `DIR/lib`, you can use the `--with-iodbc-libs=LIBDIR` option.

- If you are using unixODBC, use the `--with-unixODBC=DIR` option (case sensitive) to make **configure** look for unixODBC instead of iODBC by default, *DIR* is the directory where unixODBC is installed.

If the unixODBC headers and libraries aren't located in `DIR/include` and `DIR/lib`, use the `--with-unixODBC-includes=INCDIR` and `--with-unixODBC-libs=LIBDIR` options.

3. You might want to specify an installation prefix other than `/usr/local`. For example, to install the MyODBC drivers in `/usr/local/odbc/lib`, use the `--prefix=/usr/local/odbc` option.

The final configuration command looks something like this:

```
shell> ./configure --prefix=/usr/local \  
          --with-iodbc=/usr/local \  
          --with-mysql-path=/usr/local/mysql
```

23.1.2.4.2.2. Additional configure Options

There are a number of other options that you need, or want, to set when configuring the MyODBC driver before it is built.

- To link the driver with MySQL thread safe client libraries `libmysqlclient_r.so` or `libmysqlclient_r.a`, you must specify the following **configure** option:

```
--enable-thread-safe
```

and can be disabled (default) using

```
--disable-thread-safe
```

This option enables the building of the driver thread-safe library `libmyodbc3_r.so` from by linking with MySQL thread-safe client library `libmysqlclient_r.so` (The extensions are OS dependent).

If the compilation with the thread-safe option fails, it may be because the correct thread-libraries on the system could not be located. You should set the value of `LIBS` to point to the correct thread library for your system.

```
LIBS="-lpthread" ./configure ..
```

- You can enable or disable the shared and static versions of MyODBC using these options:

```
--enable-shared[=yes/no]  
--disable-shared  
--enable-static[=yes/no]  
--disable-static
```

- By default, all the binary distributions are built as non-debugging versions (configured with `--without-debug`).

To enable debugging information, build the driver from source distribution and use the `--with-debug` option when you run **configure**.

- This option is available only for source trees that have been obtained from the Subversion repository. This option does not apply to the packaged source distributions.

By default, the driver is built with the `--without-docs` option. If you would like the documentation to be built, then execute **configure** with:

```
--with-docs
```

23.1.2.4.2.3. Building and Compilation

To build the driver libraries, you have to just execute **make**.

```
shell> make
```

If any errors occur, correct them and continue the build process. If you aren't able to build, then send a detailed email to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com) for further assistance.

23.1.2.4.2.4. Building Shared Libraries

On most platforms, MySQL does not build or support `.so` (shared) client libraries by default. This is based on our experience of problems when building shared libraries.

In cases like this, you have to download the MySQL distribution and configure it with these options:

```
--without-server --enable-shared
```

To build shared driver libraries, you must specify the `--enable-shared` option for **configure**. By default, **configure** does not enable this option.

If you have configured with the `--disable-shared` option, you can build the `.so` file from the static libraries using the following commands:

```

shell> cd MyODBC-3.51.01
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
    $CC -bundle -flat_namespace -undefined error \
    -o .libs/libmyodbc3-3.51.01.so \
    catalog.o connect.o cursor.o dll.o error.o execute.o \
    handle.o info.o misc.o myodbc3.o options.o prepare.o \
    results.o transact.o utility.o \
    -L/usr/local/mysql/lib/mysql/ \
    -L/usr/local/iodbc/lib/ \
    -lz -lc -lmysqlclient -liodbcinst

```

Make sure to change `-liodbcinst` to `-lodbcinst` if you are using `unixODBC` instead of `iodbc`, and configure the library paths accordingly.

This builds and places the `libmyodbc3-3.51.01.so` file in the `.libs` directory. Copy this file to the MyODBC library installation directory (`/usr/local/lib` (or the `lib` directory under the installation directory that you supplied with the `-prefix`)).

```

shell> cd .libs
shell> cp libmyodbc3-3.51.01.so /usr/local/lib
shell> cd /usr/local/lib
shell> ln -s libmyodbc3-3.51.01.so libmyodbc3.so

```

To build the thread-safe driver library:

```

shell> CC=/usr/bin/gcc \
    $CC -bundle -flat_namespace -undefined error \
    -o .libs/libmyodbc3_r-3.51.01.so \
    catalog.o connect.o cursor.o dll.o error.o execute.o \
    handle.o info.o misc.o myodbc3.o options.o prepare.o \
    results.o transact.o utility.o \
    -L/usr/local/mysql/lib/mysql/ \
    -L/usr/local/iodbc/lib/ \
    -lz -lc -lmysqlclient_r -liodbcinst

```

23.1.2.4.2.5. Installing Driver Libraries

To install the driver libraries, execute the following command:

```
shell> make install
```

That command installs one of the following sets of libraries:

For MyODBC 3.51:

- `libmyodbc3.so`
- `libmyodbc3-3.51.01.so`, where 3.51.01 is the version of the driver
- `libmyodbc3.a`

For thread-safe MyODBC 3.51:

- `libmyodbc3_r.so`
- `libmyodbc3-3_r.51.01.so`
- `libmyodbc3_r.a`

For MyODBC 2.5.0:

- `libmyodbc.so`
- `libmyodbc-2.50.39.so`, where 2.50.39 is the version of the driver
- `libmyodbc.a`

For more information on build process, refer to the `INSTALL` file that comes with the source distribution. Note that if you are trying to use the **make** from Sun, you may end up with errors. On the other hand, GNU **gmake** should work fine on all platforms.

23.1.2.4.2.6. Testing MyODBC on Unix

To run the basic samples provided in the distribution with the libraries that you built, use the following command:

```
shell> make test
```

Before running the tests, create the DSN 'myodbc3' in `odbc.ini` and set the environment variable `ODBCINI` to the correct `odbc.ini` file; and MySQL server is running. You can find a sample `odbc.ini` with the driver distribution.

You can even modify the `samples/run-samples` script to pass the desired DSN,

UID, and PASSWORD values as the command-line arguments to each sample.

23.1.2.4.2.7. Building MyODBC from Source on Mac OS X

To build the driver on Mac OS X (Darwin), make use of the following **configure** example:

```
shell> ./configure --prefix=/usr/local
--with-unixODBC=/usr/local
--with-mysql-path=/usr/local/mysql
--disable-shared
--enable-gui=no
--host=powerpc-apple
```

The command assumes that the unixODBC and MySQL are installed in the default locations. If not, configure accordingly.

On Mac OS X, `--enable-shared` builds `.dylib` files by default. You can build `.so` files like this:

```
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error
-o .libs/libmyodbc3-3.51.01.so *.o
-L/usr/local/mysql/lib/
-L/usr/local/iodbc/lib
-liodbcinst -lmysqlclient -lz -lc
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error
-o .libs/libmyodbc3-3.51.01.so *.o
-L/usr/local/mysql/lib/
-L/usr/local/iodbc/lib
-liodbcinst -lmysqlclienti_r -lz -lc -lpthread
```

Make sure to change the `-liodbcinst` to `-lodbcinst` in case of using unixODBC instead of iODBC and configure the libraries path accordingly.

In Apple's version of GCC, both `cc` and `gcc` are actually symbolic links to `gcc3`.

Copy this library to the `$prefix/lib` directory and symlink to `libmyodbc3.so`.

You can cross-check the output shared-library properties using this command:

```
shell> otool -LD .libs/libmyodbc3-3.51.01.so
```

23.1.2.4.2.8. Building MyODBC from Source on HP-UX

To build the driver on HP-UX 10.x or 11.x, make use of the following **configure** example:

If using **cc**:

```
shell> CC="cc" \  
      CFLAGS="+z" \  
      LDFLAGS="-Wl,+b:-Wl,+s" \  
      ./configure --prefix=/usr/local \  
      --with-unixodbc=/usr/local \  
      --with-mysql-path=/usr/local/mysql/lib/mysql \  
      --enable-shared \  
      --enable-thread-safe
```

If using **gcc**:

```
shell> CC="gcc" \  
      LDFLAGS="-Wl,+b:-Wl,+s" \  
      ./configure --prefix=/usr/local \  
      --with-unixodbc=/usr/local \  
      --with-mysql-path=/usr/local/mysql \  
      --enable-shared \  
      --enable-thread-safe
```

Once the driver is built, cross-check its attributes using **chatr .libs/libmyodbc3.sl** to determine whether you need to have set the MySQL client library path using the `SHLIB_PATH` environment variable. For static versions, ignore all shared-library options and run **configure** with the `--disable-shared` option.

23.1.2.4.2.9. Building MyODBC from Source on AIX

To build the driver on AIX, make use of the following **configure** example:

```
shell> ./configure --prefix=/usr/local \  
      --with-unixodbc=/usr/local \  
      --with-mysql-path=/usr/local/mysql \  
      --disable-shared
```

--enable-thread-safe

NOTE: For more information about how to build and set up the static and shared libraries across the different platforms refer to '[Using static and shared libraries across platforms](#)'.

23.1.2.4.3. Installing MyODBC from the Development Source Tree

Caution: You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL Connector/ODBC up and running on your system, you should use a standard release distribution.

To be able to access the MyODBC source tree, you must have Subversion installed. Subversion is freely available from <http://subversion.tigris.org/>.

To build from the source trees, you need the following tools:

- autoconf 2.52 (or newer)
- automake 1.4 (or newer)
- libtool 1.4 (or newer)
- m4

The most recent development source tree is available from our public Subversion trees at <http://dev.mysql.com/tech-resources/sources.html>.

To checkout out the Connector/ODBC sources, change to the directory where you want the copy of the MyODBC tree to be stored, then use the following command:

```
shell> svn co http://svn.mysql.com/svnpublic/connector-odbc3
```

You should now have a copy of the entire MyODBC source tree in the directory connector-odbc3. To build from this source tree on Unix or Linux follow these steps:

```
shell> cd connector-odbc3
shell> aclocal
shell> autoheader
shell> autoconf
```

```
shell> automake;  
shell> ./configure # Add your favorite options here  
shell> make
```

For more information on how to build, refer to the `INSTALL` file located in the same directory. For more information on options to **configure**, see [Section 23.1.2.4.2.1, “Typical configure Options”](#)

When the build is done, run **make install** to install the MyODBC 3.51 driver on your system.

If you have gotten to the **make** stage and the distribution does not compile, please report it to <myodbc@lists.mysql.com>.

On Windows, make use of Windows Makefiles `WIN-Makefile` and `WIN-Makefile_debug` in building the driver. For more information, see [Section 23.1.2.4.1, “Installing MyODBC from a Source Distribution on Windows”](#).

After the initial checkout operation to get the source tree, you should run **svn update** periodically update your source according to the latest version.

23.1.3. MyODBC Configuration

Before you connect to a MySQL database using the MyODBC driver you must configure an ODBC *Data Source Name*. The DSN associates the various configuration parameters required to communicate with a database to a specific name. You use the DSN in an application to communicate with the database, rather than specifying individual parameters within the application itself. DSN information can be user specific, system specific, or provided in a special file. ODBC data source names are configured in different ways, depending on your platform and ODBC driver.

23.1.3.1. Data Source Names

A Data Source Name associates the configuration parameters for communicating with a specific database. Generally a DSN consists of the following parameters:

- Name
- Hostname

- Database Name
- Login
- Password

In addition, different ODBC drivers, including MyODBC, may accept additional driver-specific options and parameters.

There are three types of DSN:

- A *System DSN* is a global DSN definition that is available to any user and application on a particular system. A System DSN can normally only be configured by a systems administrator, or by a user who has specific permissions that let them create System DSNs.
- A *User DSN* is specific to an individual user, and can be used to store database connectivity information that the user regularly uses.
- A *File DSN* uses a simple file to define the DSN configuration. File DSNs can be shared between users and machines and are therefore more practical when installing or deploying DSN information as part of an application across many machines.

DSN information is stored in different locations depending on your platform and environment.

23.1.3.2. Configuring a MyODBC DSN on Windows

The ODBC Data Source Administrator within Windows enables you to create DSNs, check driver installation and configure ODBC systems such as tracing (used for debugging) and connection pooling.

Different editions and versions of Windows store the ODBC Data Source Administrator in different locations depending on the version of Windows that you are using.

To open the ODBC Data Source Administrator in Windows Server 2003:

1. On the Start menu, choose Administrative Tools, and then click Data Sources (ODBC).

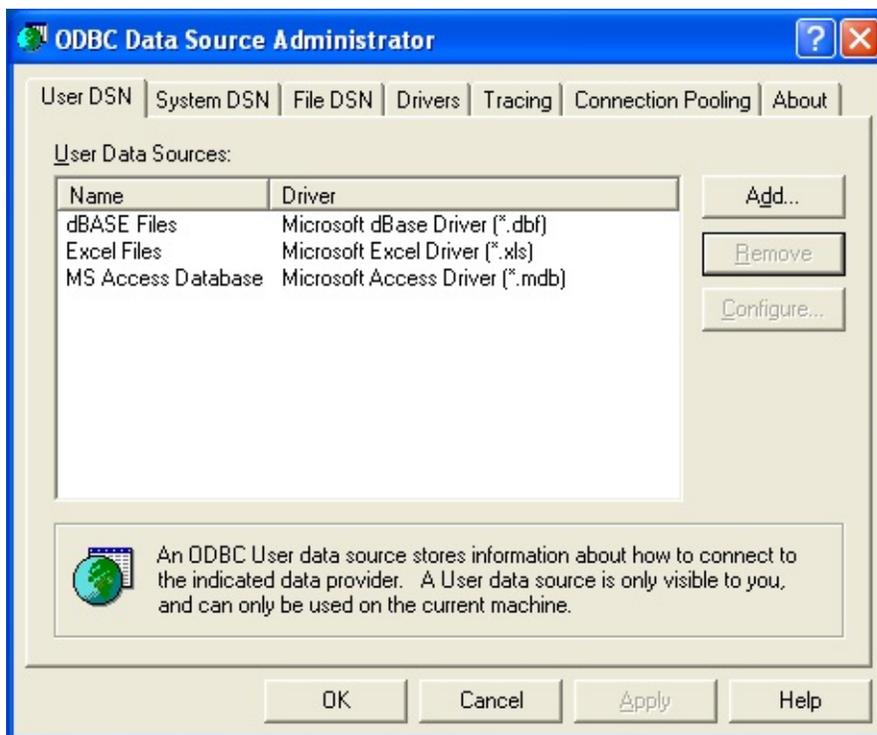
To open the ODBC Data Source Administrator in Windows 2000 Server or Windows 2000 Professional:

1. On the Start menu, choose Settings, and then click Control Panel.
2. In Control Panel, click Administrative Tools.
3. In Administrative Tools, click Data Sources (ODBC).

To open the ODBC Data Source Administrator on Windows XP:

1. On the Start menu, click Control Panel.
2. In the Control Panel when in Category View click Performance and Maintenance and then click Administrative Tools.. If you are viewing the Control Panel in Classic View, click Administrative Tools.
3. In Administrative Tools, click Data Sources (ODBC).

Irrespective of your Windows version, you should be presented the ODBC Data Source Administrator window:



Within Windows XP, you can add the Administrative Tools folder to your Start menu to make it easier to locate the ODBC Data Source Administrator. To do this:

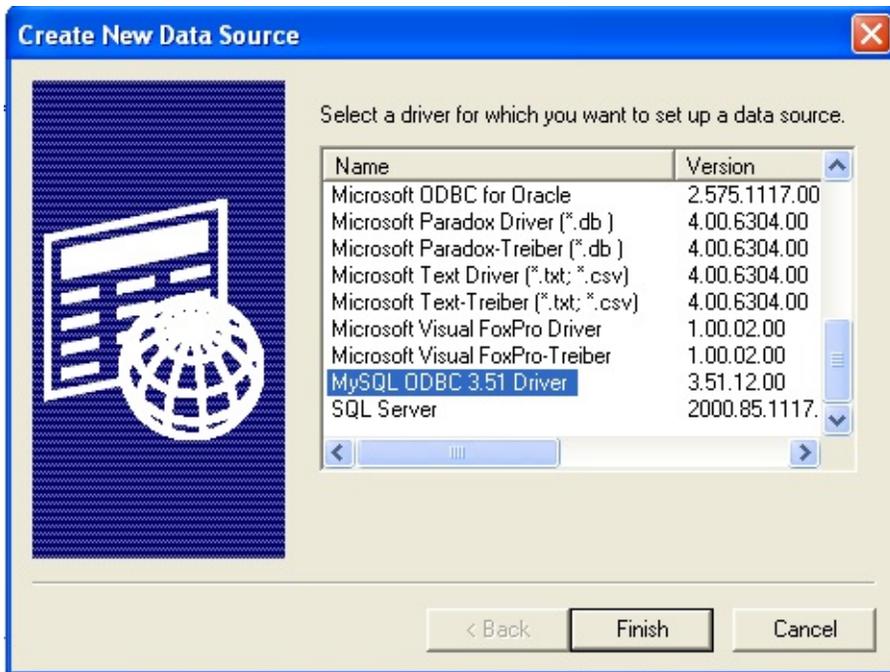
1. Right click on the Start menu.
2. Select Properties.
3. Click Customize....
4. Select the Advanced tab.
5. Within Start menu items, within the System Administrative Tools section, select Display on the All Programs menu.

Within both Windows Server 2003 and Windows XP you may want to permanently add the ODBC Data Source Administrator to your Start menu. To do this, locate the Data Sources (ODBC) icon using the methods shown, then right-click on the icon and then choose Pin to Start Menu.

23.1.3.2.1. Adding a MyODBC DSN on Windows

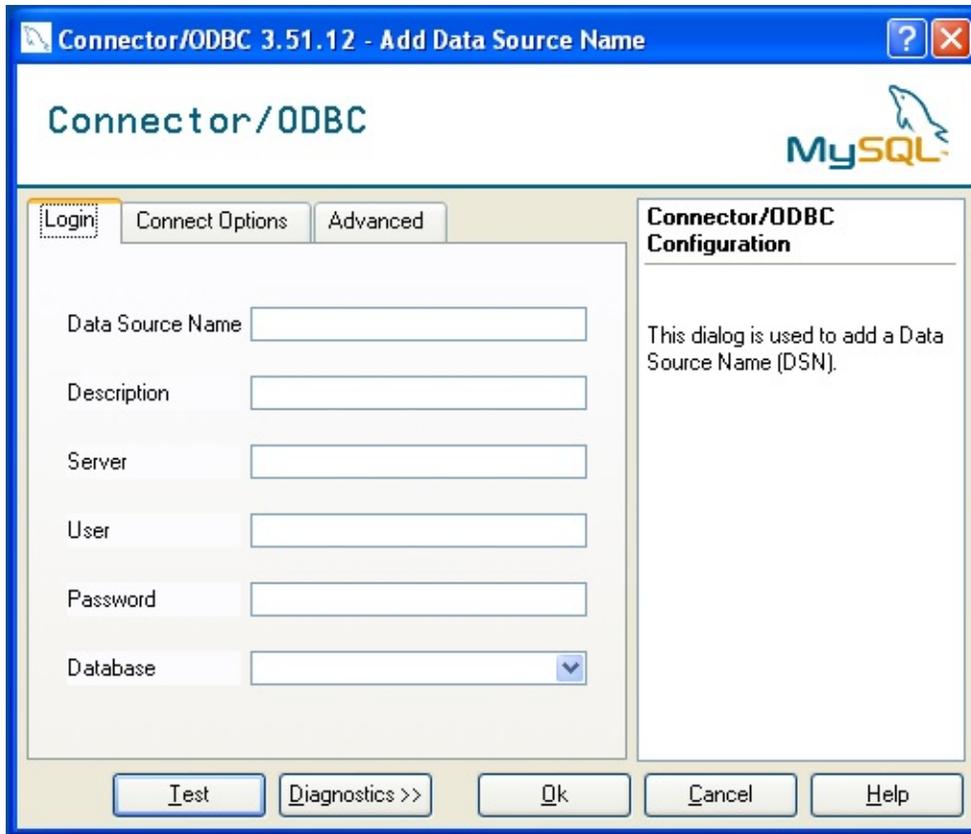
To add and configure a new MyODBC data source on Windows, use the ODBC Data Source Administrator:

1. Open the ODBC Data Source Administrator.
2. To create a System DSN (which will be available to all users) , select the System DSN tab. To create a User DSN, which will be unique only to the current user, click the Add.. button.
3. You will need to select the ODBC driver for this DSN.



Select MySQL ODBC 3.51 Driver, then click Finish.

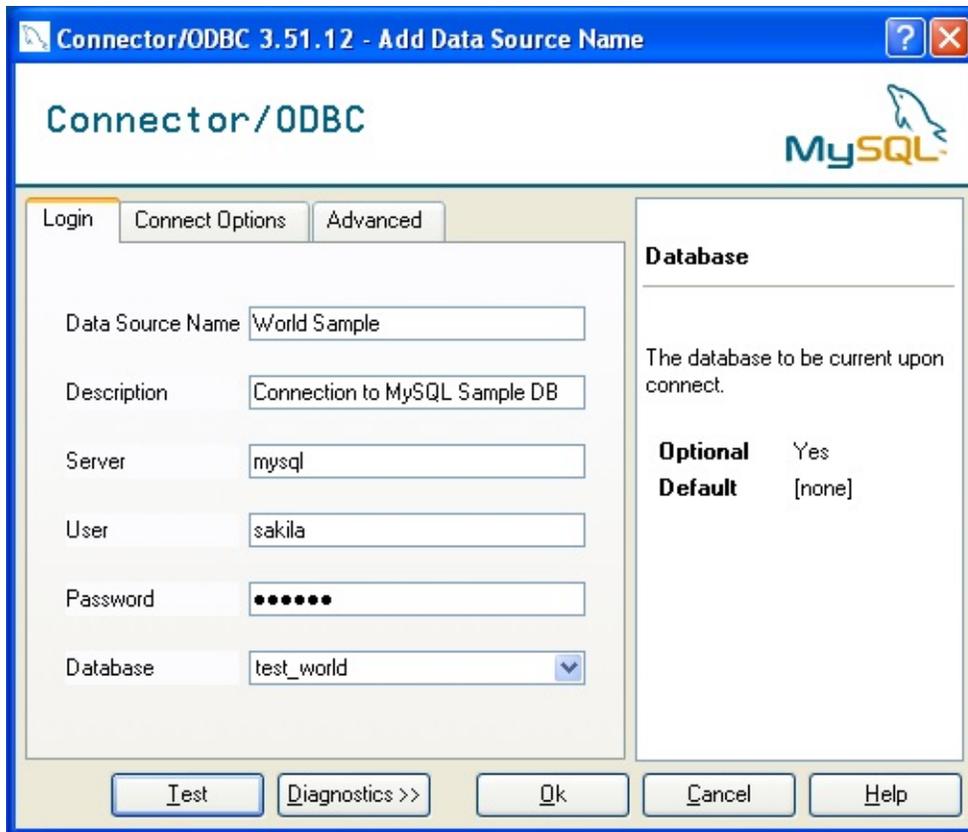
4. You now need to configure the specific fields for the DSN you are creating through the Add Data Source Name dialog.



In the Data Source Name box, enter the name of the data source you want to access. It can be any valid name that you choose.

5. In the Description box, enter some text to help identify the connection.
6. In the Server field, enter the name of the MySQL server host that you want to access. By default, it is localhost.
7. In the User field, enter the user name to use for this connection.
8. In the Password field, enter the corresponding password for this connection.
9. The Database popup should automatically populate with the list of databases that the user has permissions to access.
10. Click OK to save the DSN.

A completed DSN configuration may look like this:



23.1.3.2.2. Checking MyODBC DSN Configuration on Windows

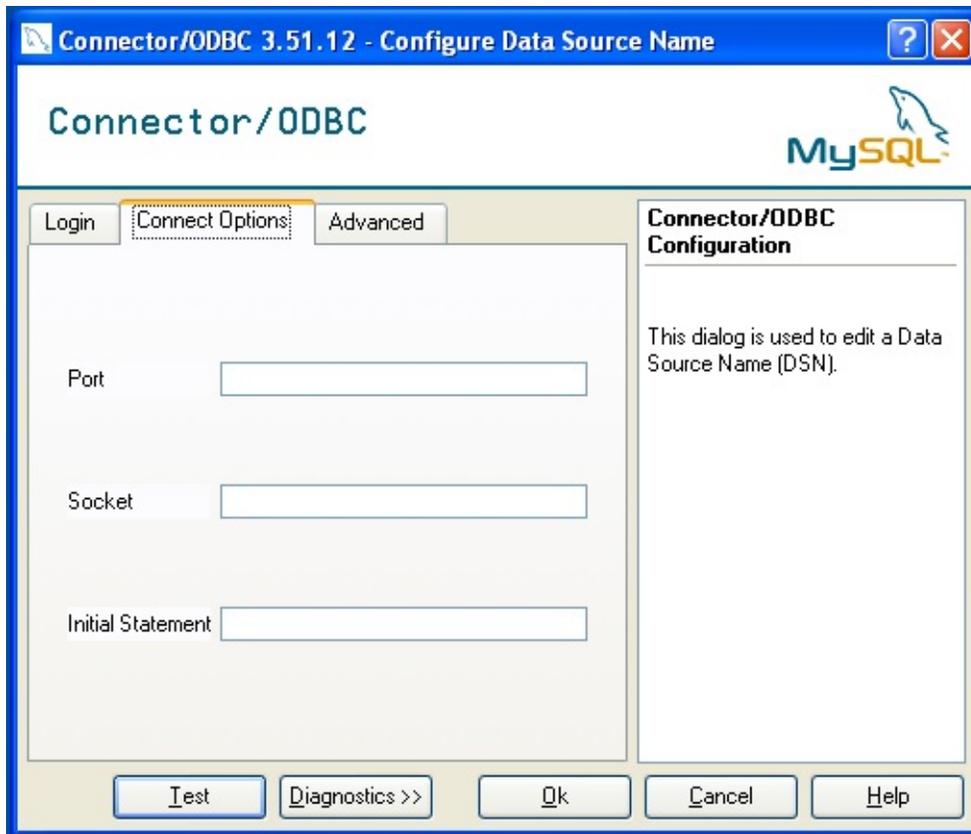
You can verify the connection using the parameters you have entered by clicking the Test button. If the connection could be made successfully, you will be notified with a Success; connection was made! dialog.

If the connection failed, you can obtain more information on the test and why it may have failed by clicking the Diagnostics... button to show additional error messages.

23.1.3.2.3. MyODBC DSN Configuration Options

You can configure a number of options for a specific DSN by using either the Connect Options or Advanced tabs in the DSN configuration dialog.

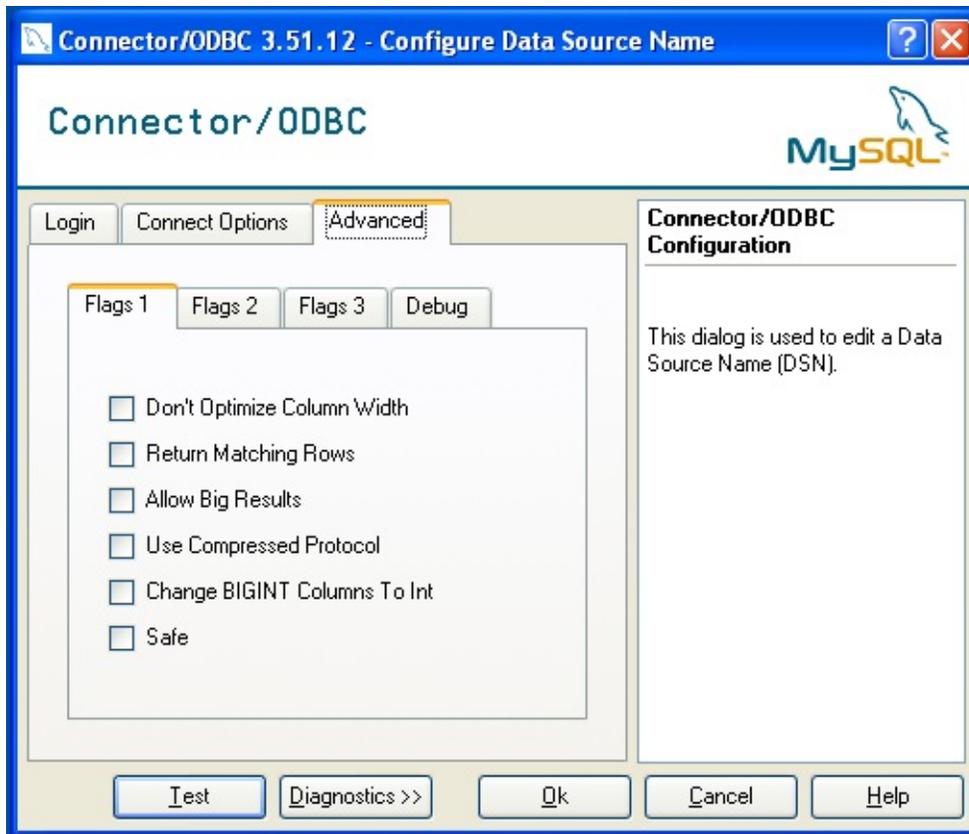
The Connection Options dialog can be seen below.



The three options you can configure are:

- Port sets the TCP/IP port number to use when communicating with MySQL. Communication with MySQL uses port 3306 by default. If your server is configured to use a different TCP/IP port, you must specify that port number here.
- Socket sets the name or location of a specific socket or Windows pipe to use when communicating with MySQL.
- Initial Statement defines an SQL statement that will be executed when the connection to MySQL is opened. You can use this to set MySQL options for your connection, such as setting the default character set or database to use during your connection.

The Advanced tab enables you to configure MyODBC connection parameters. Refer to [Section 23.1.3.5, “MyODBC Connection Parameters”](#), for information about the meaning of these options.



23.1.3.2.4. Errors and Debugging

This section answers MyODBC connection-related questions.

- **While configuring a MyODBC DSN, a Could Not Load Translator or Setup Library error occurs**

For more information, refer to [MS KnowledgeBase Article\(Q260558\)](#). Also, make sure you have the latest valid `ct13d32.dll` in your system directory.

- On Windows, the default `myodbc3.dll` is compiled for optimal performance. If you want to debug MyODBC 3.51 (for example, to enable tracing), you should instead use `myodbc3d.dll`. To install this file, copy `myodbc3d.dll` over the installed `myodbc3.dll` file. Make sure to revert back to the release version of the driver DLL once you are done with the debugging because the debug version may cause performance issues. Note that the `myodbc3d.dll` isn't included in MyODBC 3.51.07 through 3.51.11. If you are using one of these versions, you should copy that DLL from a

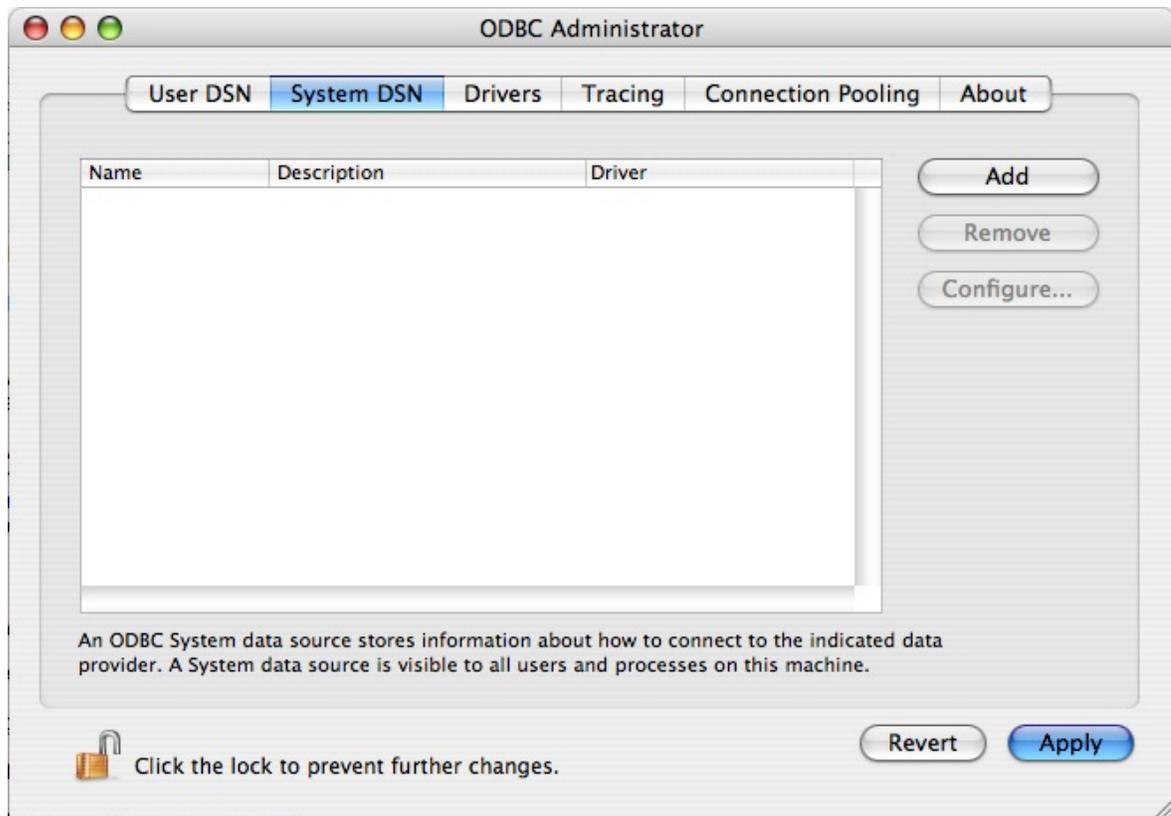
previous version (for example, 3.51.06).

For MyODBC 2.50, `myodbc.d11` and `myodbcd.d11` are used instead.

23.1.3.3. Configuring a MyODBC DSN on Mac OS X

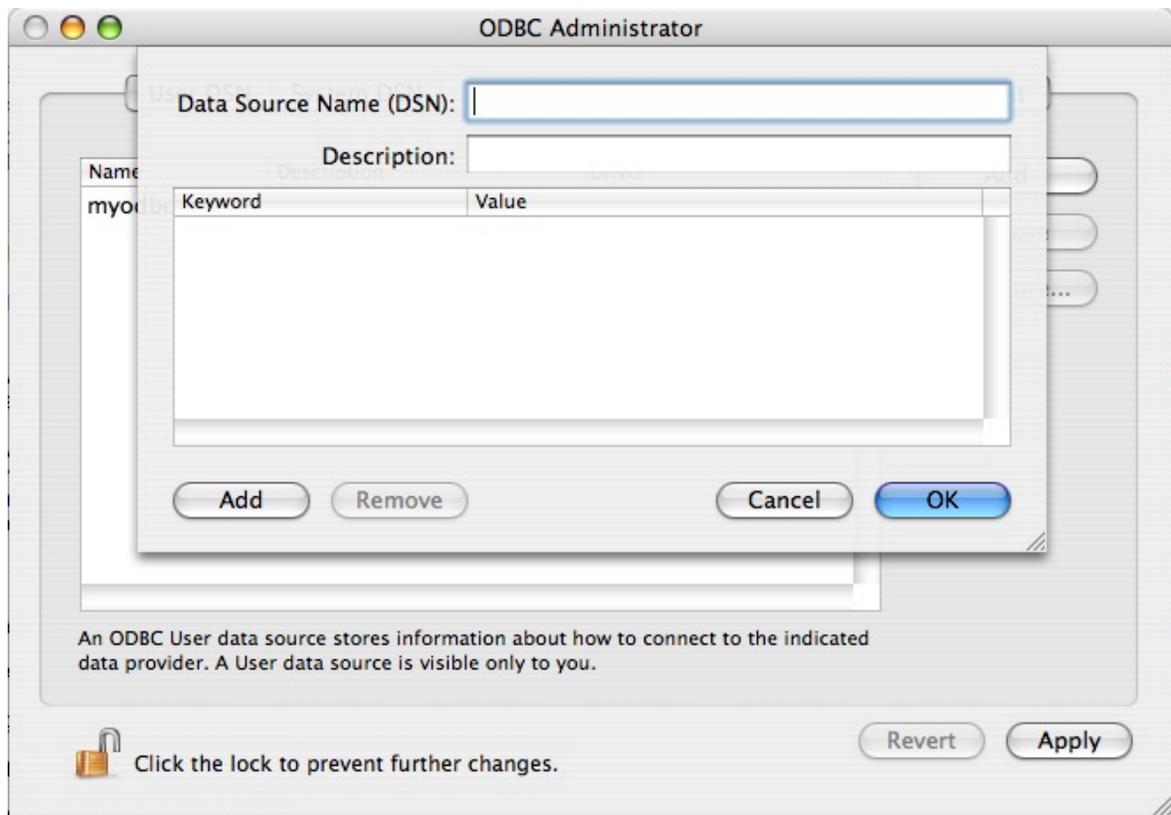
To configure a DSN on Mac OS X you should use the ODBC Administrator. If you have Mac OS X 10.2 or earlier, refer to [Section 23.1.3.4, “Configuring a MyODBC DSN on Unix”](#). Select whether you want to create a User DSN or a System DSN. If you want to add a System DSN, you may need to authenticate with the system. You must click the padlock and enter a user and password with administrator privileges.

1. Open the ODBC Administrator from the `Utilities` folder in the `Applications` folder.



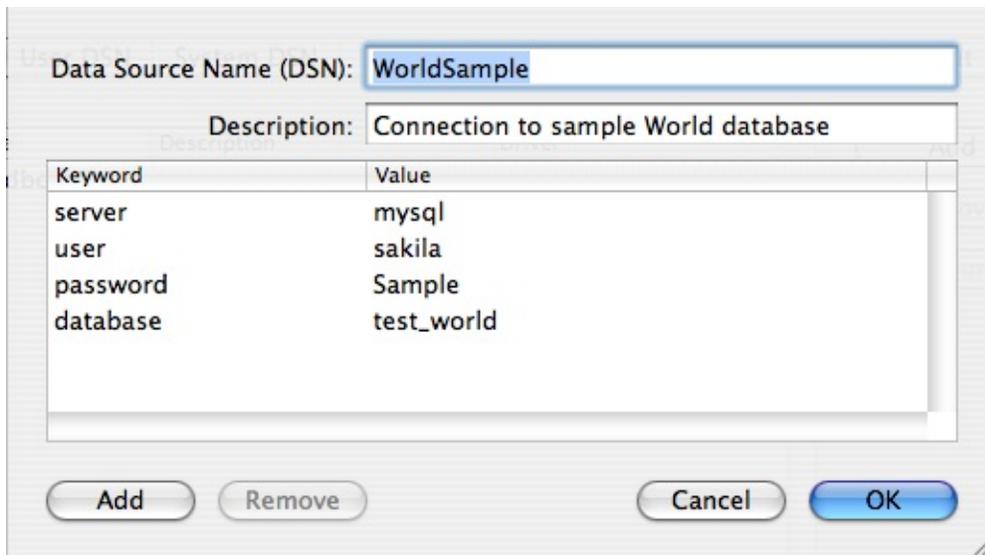
2. On the User DSN or System DSN panel, click Add.
3. Select the MyODBC driver and click OK.

4. You will be presented with the Data Source Name dialog. Enter The Data Source Name and an optional Description for the DSN.



5. Click Add to add a new keyword/value pair to the panel. You should configure at least four pairs to specify the server, username, password and database connection parameters. See [Section 23.1.3.5, “MyODBC Connection Parameters”](#).
6. Click OK to add the DSN to the list of configured data source names.

A completed DSN configuration may look like this:



You can configure additional ODBC options to your DSN by adding further keyword/value pairs and setting the corresponding values. See [Section 23.1.3.5, “MyODBC Connection Parameters”](#).

23.1.3.4. Configuring a MyODBC DSN on Unix

On Unix, you configure DSN entries directly in the `odbc.ini` file. Here is a typical `odbc.ini` file that configures `myodbc` and `myodbc3` as the DSN names for MyODBC 2.50 and MyODBC 3.51, respectively:

```

;
; odbc.ini configuration for MyODBC and MyODBC 3.51 drivers
;

[ODBC Data Sources]
myodbc      = MyODBC 2.50 Driver DSN
myodbc3     = MyODBC 3.51 Driver DSN

[myodbc]
Driver      = /usr/local/lib/libmyodbc.so
Description = MyODBC 2.50 Driver DSN
SERVER     = localhost
PORT      =
USER      = root
Password   =
Database   = test
OPTION    = 3
SOCKET     =

```

```
[myodbc3]
Driver      = /usr/local/lib/libmyodbc3.so
Description = MyODBC 3.51 Driver DSN
SERVER     = localhost
PORT       =
USER       = root
Password   =
Database   = test
OPTION     = 3
SOCKET     =
```

```
[Default]
Driver      = /usr/local/lib/libmyodbc3.so
Description = MyODBC 3.51 Driver DSN
SERVER     = localhost
PORT       =
USER       = root
Password   =
Database   = test
OPTION     = 3
SOCKET     =
```

Refer to the [Section 23.1.3.5, “MyODBC Connection Parameters”](#), for the list of connection parameters that can be supplied.

Note: If you are using unixODBC, you can use the following tools to set up the DSN:

- ODBCConfig GUI tool([HOWTO: ODBCConfig](#))
- odbcinst

In some cases when using unixODBC, you might get this error:

```
Data source name not found and no default driver specified
```

If this happens, make sure the ODBCINI and ODBCSYSINI environment variables are pointing to the right `odbc.ini` file. For example, if your `odbc.ini` file is located in `/usr/local/etc`, set the environment variables like this:

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

23.1.3.5. MyODBC Connection Parameters

You can specify the parameters in the following tables for MyODBC when configuring a DSN. Users on Windows can use the Options and Advanced panels when configuring a DSN to set these parameters; see the table for information on which options relate to which fields and checkboxes. On Unix and Mac OS X, use the parameter name and value as the keyword/value pair in the DSN configuration. Alternatively, you can set these parameters within the InConnectionString argument in the SQLDriverConnect() call.

Parameter	Default Value	Comment
user	ODBC (on Windows)	The username used to connect to MySQL.
server	localhost	The hostname of the MySQL server.
database		The default database.
option	0	Options that specify how MyODBC should work. See below.
port	3306	The TCP/IP port to use if server is not localhost.
stmt		A statement to execute when connecting to MySQL.
password		The password for the user account on server.
socket		The Unix socket file or Windows named pipe to connect to if server is localhost.

The option argument is used to tell MyODBC that the client isn't 100% ODBC compliant. On Windows, you normally select options by toggling the checkboxes in the connection screen, but you can also select them in the option argument. The following options are listed in the order in which they appear in the MyODBC connect screen:

Value	Windows Checkbox	Description
1	Don't Optimized Column Width	The client can't handle that MyODBC returns the real width of a column.
2	Return Matching Rows	The client can't handle that MySQL returns the true value of affected rows. If this flag is set, MySQL returns "found rows" instead. You must have MySQL 3.21.14 or newer to get this to work.

4	Trace Driver Calls To myodbc.log	Make a debug log in c:\myodbc.log on Windows, or /tmp/myodbc.log on Unix variants.
8	Allow Big Results	Don't set any packet limit for results and parameters.
16	Don't Prompt Upon Connect	Don't prompt for questions even if driver would like to prompt.
32	Enable Dynamic Cursor	Enable or disable the dynamic cursor support. (Not allowed in MyODBC 2.50.)
64	Ignore # in Table Name	Ignore use of database name in db_name.tbl_name.col_name.
128	User Manager Cursors	Force use of ODBC manager cursors (experimental).
256	Don't Use Set Locale	Disable the use of extended fetch (experimental).
512	Pad Char To Full Length	Pad CHAR columns to full column length.
1024	Return Table Names for SQLDescribeCol	SQLDescribeCol() returns fully qualified column names.
2048	Use Compressed Protocol	Use the compressed client/server protocol.
4096	Ignore Space After Function Names	Tell server to ignore space after function name and before '(' (needed by PowerBuilder). This makes all function names keywords.
8192	Force Use of Named Pipes	Connect with named pipes to a mysqld server running on NT.
16384	Change BIGINT Columns to Int	Change BIGINT columns to INT columns (some applications can't handle BIGINT).
32768	No Catalog (exp)	Return 'user' as Table_qualifier and Table_owner from SQLTables (experimental).
65536	Read Options From my.cnf	Read parameters from the [client] and [odbc] groups from my.cnf.
131072	Safe	Add some extra safety checks (should not be needed but...).

262144	Disable transaction	Disable transactions.
524288	Save queries to myodbc.sql	Enable query logging to c:\myodbc.sql(/tmp/myodbc.sql) file. (Enabled only in debug mode.)
1048576	Don't Cache Result (forward only cursors)	Do not cache the results locally in the driver, instead read from server (mysql_use_result()). This works only for forward-only cursors. This option is very important in dealing with large tables when you don't want the driver to cache the entire result set.
2097152	Force Use Of Forward Only Cursors	Force the use of Forward-only cursor type. In case of applications setting the default static/dynamic cursor type, and one wants the driver to use non-cache result sets, then this option ensures the forward-only cursor behavior.

To select multiple options, add together their values. For example, setting option to 12 (4+8) gives you debugging without packet limits.

The following table shows some recommended option values for various configurations:

Configuration	Option Value
Microsoft Access, Visual Basic	3
Driver trace generation (Debug mode)	4
Microsoft Access (with improved DELETE queries)	35
Large tables with too many rows	2049
Sybase PowerBuilder	135168
Query log generation (Debug mode)	524288
Generate driver trace as well as query log (Debug mode)	524292
Large tables with no-cache results	3145731

23.1.3.6. Connecting Without a Predefined DSN

You can connect to the MySQL server using `SQLDriverConnect`, by specifying the `DRIVER` name field. Here are the connection strings for MyODBC using DSN-Less connections:

For MyODBC 2.50:

```
ConnectionString = "DRIVER={MySQL};\  
SERVER=localhost;\  
DATABASE=test;\  
USER=venu;\  
PASSWORD=venu;\  
OPTION=3;"
```

For MyODBC 3.51:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\  
SERVER=localhost;\  
DATABASE=test;\  
USER=venu;\  
PASSWORD=venu;\  
OPTION=3;"
```

If your programming language converts backslash followed by whitespace to a space, it is preferable to specify the connection string as a single long string, or to use a concatenation of multiple strings that does not add spaces in between. For example:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"  
"SERVER=localhost;"  
"DATABASE=test;"  
"USER=venu;"  
"PASSWORD=venu;"  
"OPTION=3;"
```

Note. Note that on Mac OS X you may need to specify the full path to the MyODBC driver library.

Refer to the [Section 23.1.3.5, “MyODBC Connection Parameters”](#), for the list of connection parameters that can be supplied.

23.1.3.7. ODBC Connection Pooling

Connection pooling enables the ODBC driver to re-use existing connections to a

given database from a pool of connections, instead of opening a new connection each time the database is accessed. By enabling connection pooling you can improve the overall performance of your application by lowering the time taken to open a connection to a database in the connection pool.

For more information about connection pooling:

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169470>.

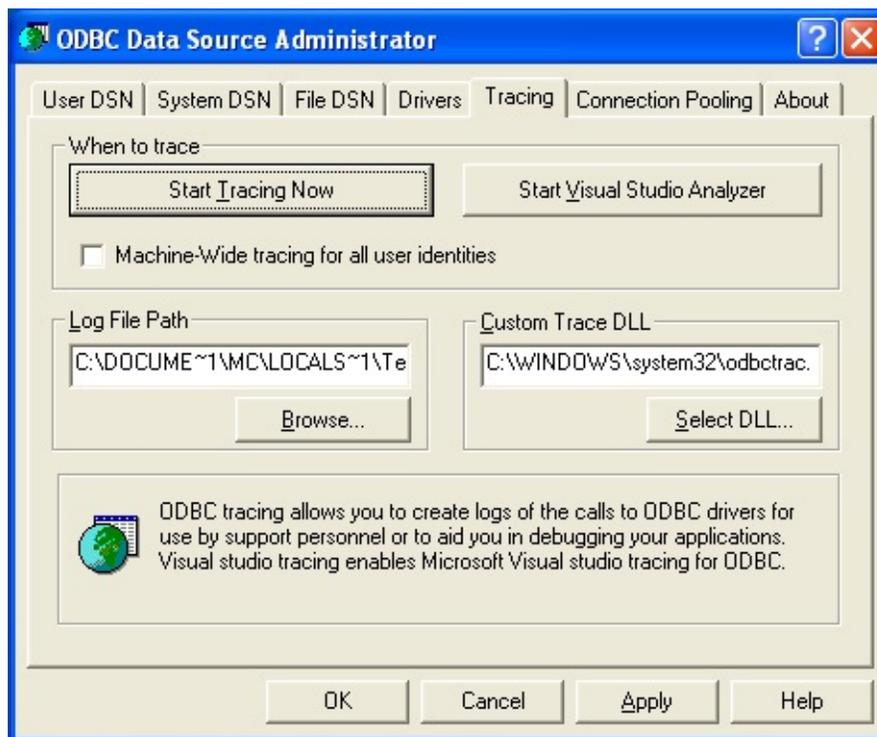
23.1.3.8. Getting an ODBC Trace File

If you encounter difficulties or problems with MyODBC, you should start by making a log file from the ODBC Manager and MyODBC. This is called *tracing*, and is enabled through the ODBC Manager. The procedure for this differs for Windows, Mac OS X and Unix.

23.1.3.8.1. Enabling ODBC Tracing on Windows

To enable the trace option on Windows:

1. The Tracing tab of the ODBC Data Source Administrator dialog box enables you to configure the way ODBC function calls are traced.

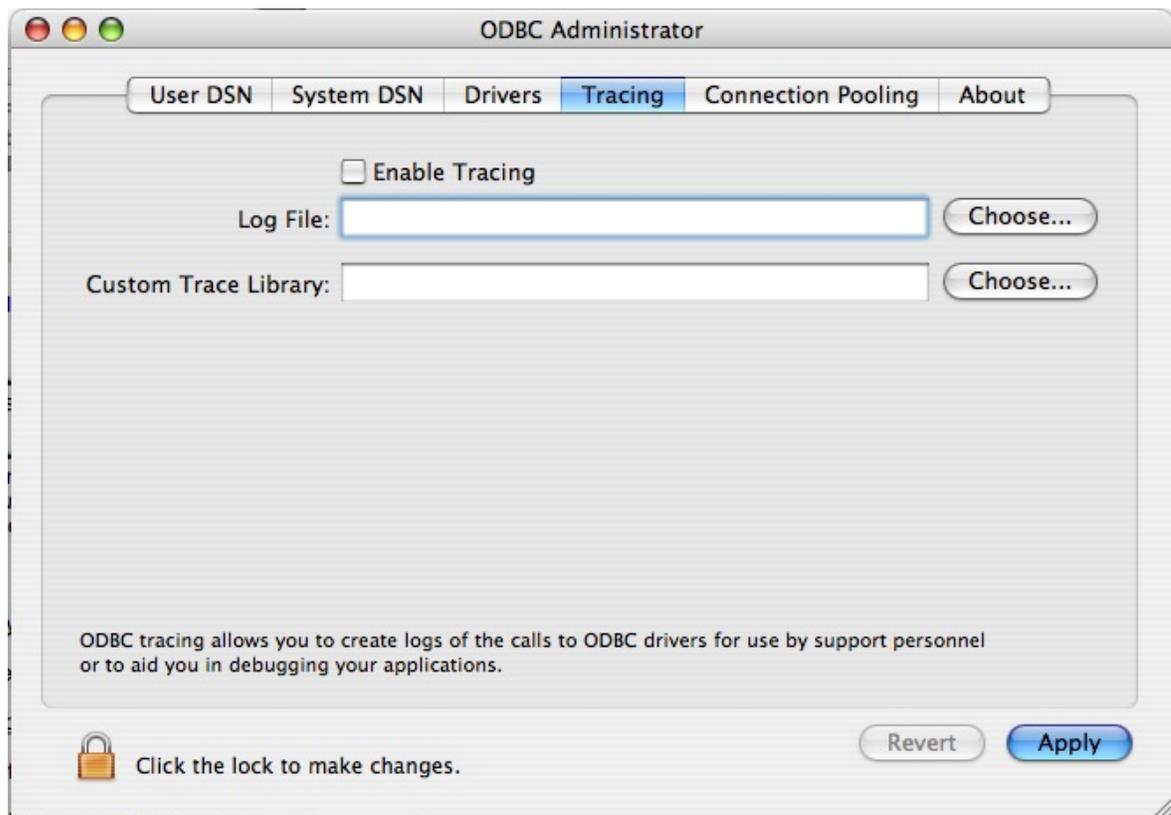


2. When you activate tracing from the Tracing tab, the Driver Manager logs all ODBC function calls for all subsequently run applications.
3. ODBC function calls from applications running before tracing is activated are not logged. ODBC function calls are recorded in a log file you specify.
4. Tracing ceases only after you click Stop Tracing Now. Remember that while tracing is on, the log file continues to increase in size and that tracing affects the performance of all your ODBC applications.

23.1.3.8.2. Enabling ODBC Tracing on Mac OS X

To enable the trace option on Mac OS X 10.3 or later you should use the Tracing tab within ODBC Administrator .

1. Open the ODBC Administrator.
2. Select the Tracing tab.



3. Select the Enable Tracing checkbox.

4. Enter the location where you want to save the Tracing log. If you want to append information to an existing log file, click the Choose... button.

23.1.3.8.3. Enabling ODBC Tracing on Unix

To enable the trace option on Mac OS X 10.2 (or earlier) or Unix you must add the trace option to the ODBC configuration:

1. On Unix, you need to explicitly set the Trace option in the ODBC.INI file.

Set the tracing ON or OFF by using TraceFile and Trace parameters in odbc.ini as shown below:

```
TraceFile = /tmp/odbc.trace
Trace     = 1
```

TraceFile specifies the name and full path of the trace file and Trace is set to ON or OFF. You can also use 1 or YES for ON and 0 or NO for OFF. If you are using **ODBCConfig** from unixODBC, then follow the instructions for tracing unixODBC calls at [HOWTO-ODBCConfig](#).

23.1.3.8.4. Enabling a MyODBC Log

To generate a MyODBC log, do the following:

1. Within Windows, enable the Trace MyODBC option flag in the MyODBC connect/configure screen. The log is written to file c:\myodbc.log. If the trace option is not remembered when you are going back to the above screen, it means that you are not using the myodbcd.dll driver, see [Section 23.1.3.2.4, “Errors and Debugging”](#).

On Mac OS X, Unix, or if you are using DSN-Less connection, then you need to supply OPTION=4 in the connection string or set the corresponding keyword/value pair in the DSN.

2. Start your application and try to get it to fail. Then check the MyODBC trace file to find out what could be wrong.

If you need help determining what is wrong, see [Section 23.1.7.1, “MyODBC Community Support”](#).

23.1.4. MyODBC Examples

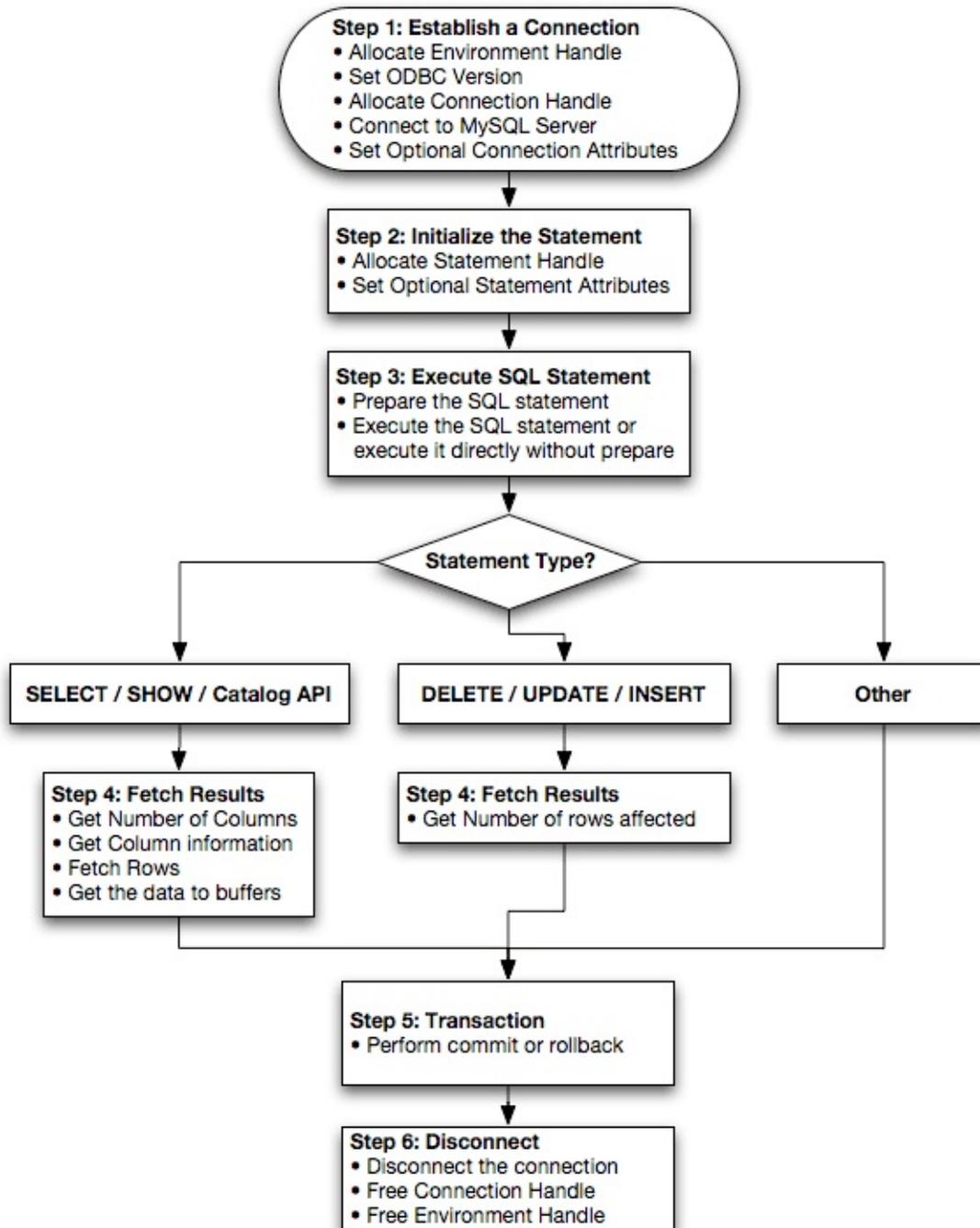
Once you have configured a DSN to provide access to a database, how you access and use that connection is dependent on the application or programming language. As ODBC is a standardized interface, any application or language that supports ODBC can use the DSN and connect to the configured database.

23.1.4.1. Basic MyODBC Application Steps

Interacting with a MySQL server from an applications using the MyODBC typically involves the following operations:

- Configure the MyODBC DSN
- Connect to MySQL server
- Initialization operations
- Execute SQL statements
- Retrieve results
- Perform Transactions
- Disconnect from the server

Most applications use some variation of these steps. The basic application steps are shown in the following diagram:



23.1.4.2. Step-by-step Guide to Connecting to a MySQL Database through MyODBC

A typical installation situation where you would install MyODBC is when you want to access a database on a Linux or Unix host from a Windows machine.

As an example of the process required to set up access between two machines, the steps below take you through the basic steps. These instructions assume that you want to connect to system ALPHA from system BETA with a username and password of myuser and mypassword.

On system ALPHA (the MySQL server) follow these steps:

1. Start the MySQL server.
2. Use GRANT to set up an account with a username of myuser that can connect from system BETA using a password of myuser to the database test:

```
GRANT ALL ON test.* to 'myuser'@'BETA' IDENTIFIED BY 'mypassword'
```

For more information about MySQL privileges, refer to [Section 5.9, “MySQL User Account Management”](#).

On system BETA (the MyODBC client), follow these steps:

1. Configure a MyODBC DSN using parameters that match the server, database and authentication information that you have just configured on system ALPHA.

Parameter	Value	Comment
DSN	remote_test	A name to identify the connection.
SERVER	ALPHA	The address of the remote server.
DATABASE	test	The name of the default database.
USER	myuser	The username configured for access to this database.
PASSWORD	mypassword	The password for myuser.

2. Using an ODBC-capable application, such as Microsoft Office, connect to the MySQL server using the DSN you have just created. If the connection fails, use tracing to examine the connection process. See [Section 23.1.3.8, “Getting an ODBC Trace File”](#), for more information.

23.1.4.3. MyODBC and Third-Party ODBC Tools

Once you have configured your MyODBC DSN, you can access your MySQL database through any application that supports the ODBC interface, including programming languages and third-party applications. This section contains guides and help on using MyODBC with various ODBC-compatible tools and applications, including Microsoft Word, Microsoft Excel and Adobe/Macromedia ColdFusion.

23.1.4.3.1. Applications Tested with MyODBC

MyODBC has been tested with the following applications:

Publisher	Application	Notes
Adobe	ColdFusion	Formerly Macromedia ColdFusion
Borland	C++ Builder	
	Builder 4	
	Delphi	
Business Objects	Crystal Reports	
Claris	Filemaker Pro	
Corel	Paradox	
Computer Associates	Visual Objects	Also known as CAVO
	AllFusion ERwin Data Modeler	
Gupta	Team Developer	Previously known as Centura Team Developer; Gupta SQL/Windows
Gensym	G2-ODBC Bridge	
Inline	iHTML	
Lotus	Notes	Versions 4.5 and 4.6
Microsoft	Access	
	Excel	
	Visio Enterprise	
	Visual C++	
	Visual Basic	
	ODBC.NET	Using C#, Visual Basic, C++

	FoxPro	
	Visual Interdev	
OpenOffice.org	OpenOffice.org	
Perl	DBD::ODBC	
Pervasive Software	DataJunction	
Sambar Technologies	Sambar Server	
SPSS	SPSS	
SoftVelocity	Clarion	
SQLExpress	SQLExpress for Xbase++	
Sun	StarOffice	
SunSystems	Vision	
Sybase	PowerBuilder	
	PowerDesigner	
theKompany.com	Data Architect	

If you know of any other applications that work with MyODBC, please send mail to <myodbc@lists.mysql.com> about them.

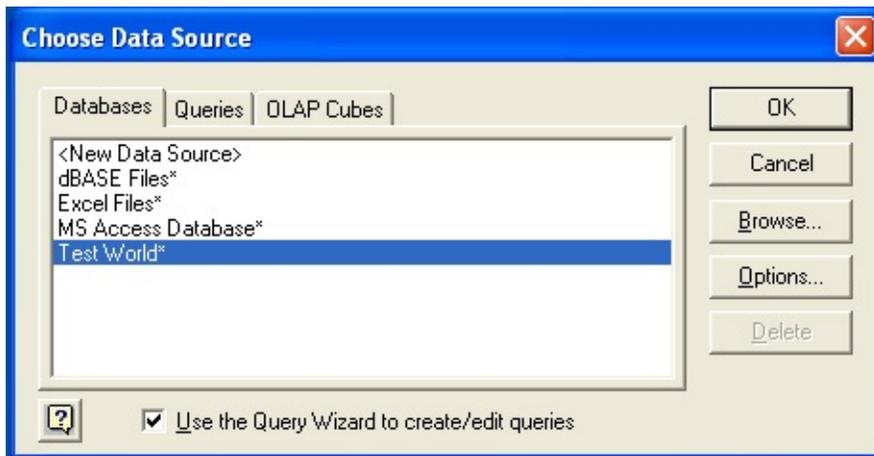
23.1.4.3.2. Using MyODBC with Microsoft Word or Excel

You can use Microsoft Word and Microsoft Excel to access information from a MySQL database using MyODBC. Within Microsoft Word, this facility is most useful when importing data for mailmerge, or for tables and data to be included in reports. Within Microsoft Excel, you can execute queries on your MySQL server and import the data directly into an Excel Worksheet, presenting the data as a series of rows and columns.

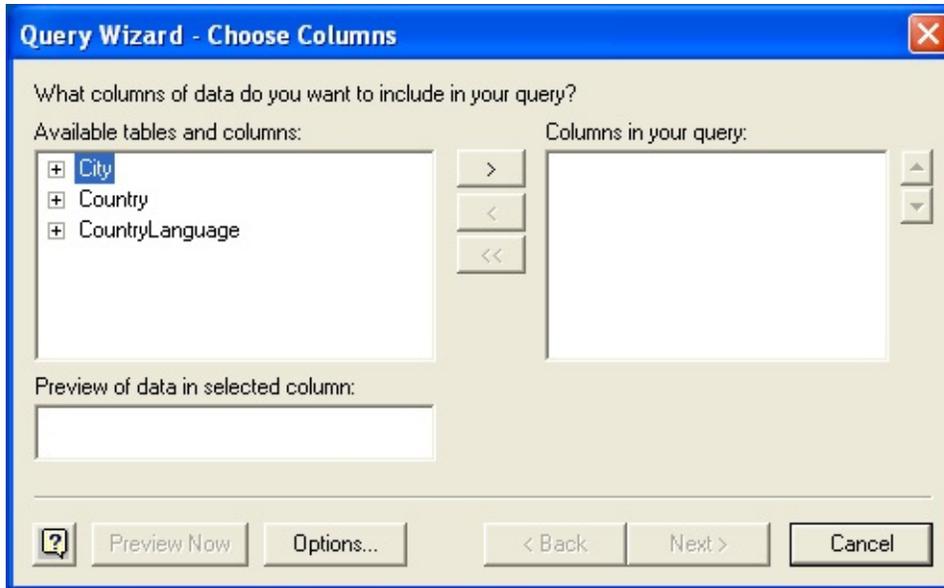
With both applications, data is accessed and imported into the application using Microsoft Query, which enables you to execute a query through an ODBC source. You use Microsoft Query to build the SQL statement to be executed, selecting the tables, fields, selection criteria and sort order. For example, to insert information from a table in the World test database into an Excel spreadsheet,

using the DSN samples shown in [Section 23.1.3, “MyODBC Configuration”](#):

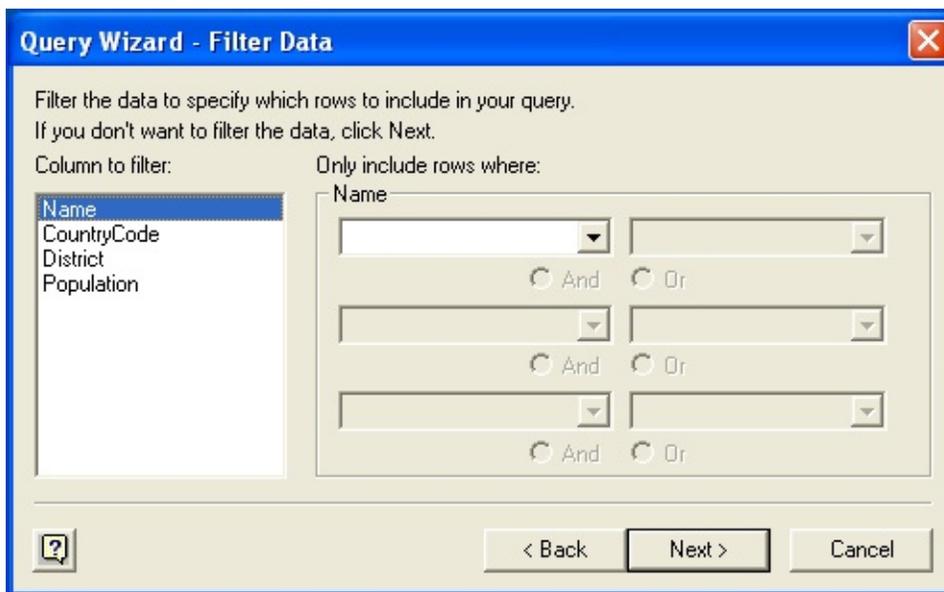
1. Create a new Worksheet.
2. From the Data menu, choose Import External Data, and then select New Database Query.
3. Microsoft Query will start. First, you need to choose the data source, by selecting an existing Data Source Name.



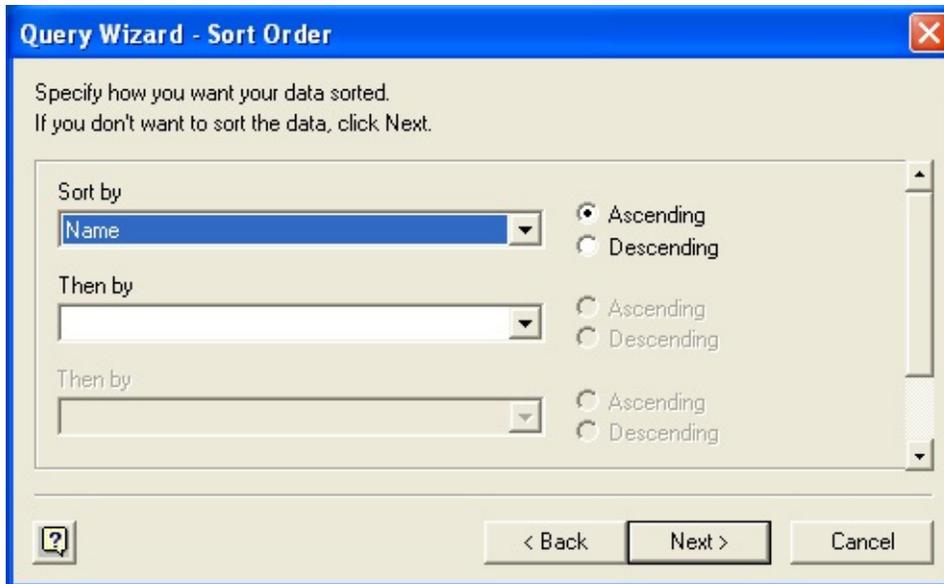
4. Within the Query wizard, you must choose the columns that you want to import. The list of tables available to the user configured through the DSN is shown on the left, the columns that will be added to your query are shown on the right. The columns you choose are equivalent to those in the first section of a SELECT query. Click Next to continue.



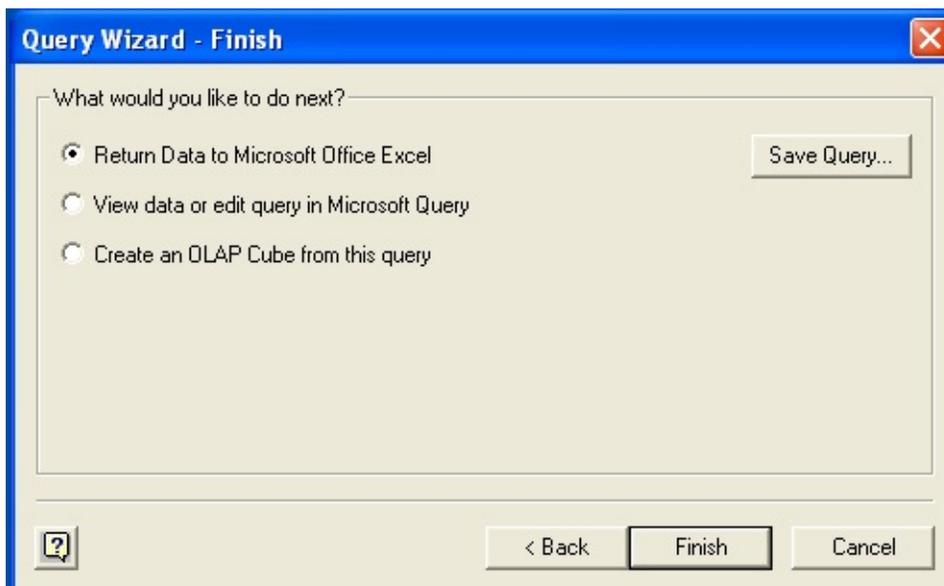
5. You can filter rows from the query (the equivalent of a `WHERE` clause) using the `Filter Data` dialog. Click `Next` to continue.



6. Select an (optional) sort order for the data. This is equivalent to using a `ORDER BY` clause in your SQL query. You can select up to three fields for sorting the information returned by the query. Click `Next` to continue.



7. Select the destination for your query. You can select to return the data Microsoft Excel, where you can choose a worksheet and cell where the data will be inserted; you can continue to view the query and results within Microsoft Query, where you can edit the SQL query and further filter and sort the information returned; or you can create an OLAP Cube from the query, which can then be used directly within Microsoft Excel. Click Finish.



The same process can be used to import data into a Word document, where the data will be inserted as a table. This can be used for mail merge purposes (where

the field data is read from a Word table), or where you want to include data and reports within a report or other document.

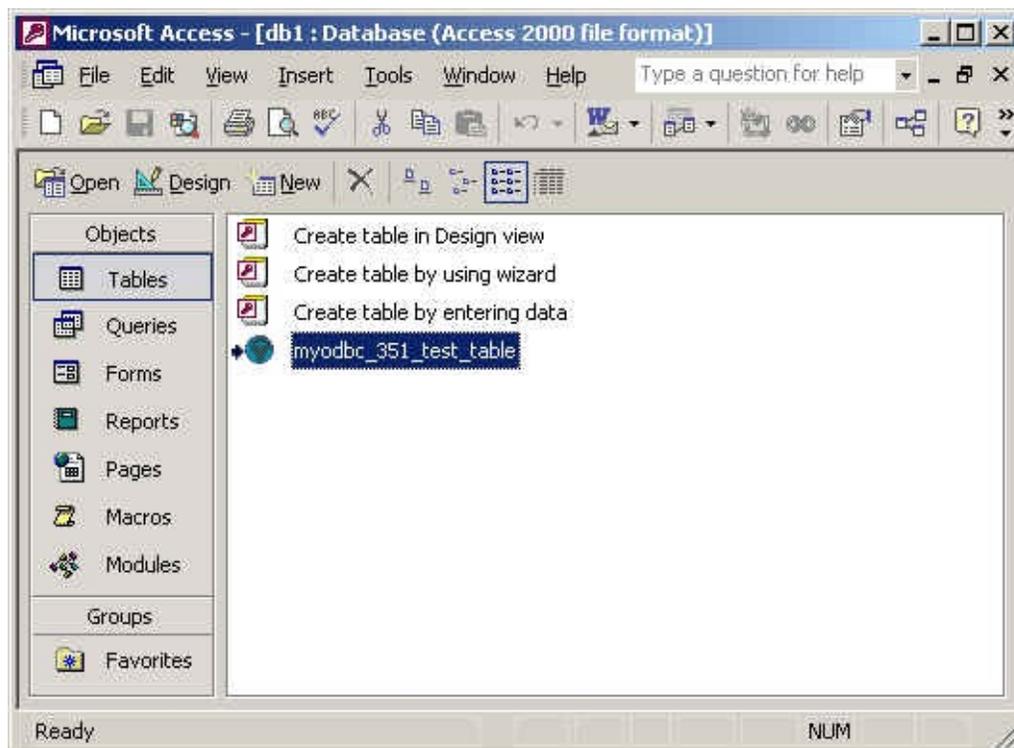
23.1.4.3.3. Using MyODBC and Microsoft Access

You can use MySQL database with Microsoft Access using MyODBC. The MySQL database can be used as an import source, an export source, or as a linked table for direct use within an Access application, so you can use Access as the front-end interface to a MySQL database.

23.1.4.3.3.1. Exporting Access Data to MySQL

To export a table of data from an Access database to MySQL, follow these instructions:

1. When you open an Access database or an Access project, a Database window appears. It displays shortcuts for creating new database objects and opening existing objects.



2. Click the name of the table or query you want to export, and then in the File menu, select Export.

3. In the Export Object Type Object name To dialog box, in the Save As Type box, select ODBC Databases () as shown here:



4. In the Export dialog box, enter a name for the file (or use the suggested name), and then select OK.
5. The Select Data Source dialog box is displayed; it lists the defined data sources for any ODBC drivers installed on your computer. Click either the File Data Source or Machine Data Source tab, and then double-click the MyODBC or MyODBC 3.51 data source that you want to export to. To define a new data source for MyODBC, please [Section 23.1.3.2, “Configuring a MyODBC DSN on Windows”](#).

Microsoft Access connects to the MySQL Server through this data source and exports new tables and or data.

23.1.4.3.3.2. Importing MySQL Data to Access

To import or link a table or tables from MySQL to Access, follow these instructions:

1. Open a database, or switch to the Database window for the open database.
2. To import tables, on the File menu, point to Get External Data, and then click Import. To link tables, on the File menu, point to Get External Data, and then click Link Tables.
3. In the Import (or Link) dialog box, in the Files Of Type box, select ODBC Databases (). The Select Data Source dialog box lists the defined data sources The Select Data Source dialog box is displayed; it lists the defined data source names.
4. If the ODBC data source that you selected requires you to log on, enter

your login ID and password (additional information might also be required), and then click OK.

5. Microsoft Access connects to the MySQL server through ODBC data source and displays the list of tables that you can import or link.
6. Click each table that you want to import or link, and then click OK. If you're linking a table and it doesn't have an index that uniquely identifies each record, Microsoft Access displays a list of the fields in the linked table. Click a field or a combination of fields that uniquely identifies each record, and then click OK.

23.1.4.3.3.3. Linking MySQL Data to Access Tables

Use the following procedure to view or to refresh links when the structure or location of a linked table has changed. The Linked Table Manager lists the paths to all currently linked tables.

To view or refresh links:

1. Open the database that contains links to tables.
2. On the Tools menu, point to Add-ins (Database Utilities in Access 2000 or newer), and then click Linked Table Manager.
3. Select the check box for the tables whose links you want to refresh.
4. Click OK to refresh the links.

Microsoft Access confirms a successful refresh or, if the table wasn't found, displays the Select New Location of <table name> dialog box in which you can specify its the table's new location. If several selected tables have moved to the new location that you specify, the Linked Table Manager searches that location for all selected tables, and updates all links in one step.

To change the path for a set of linked tables:

1. Open the database that contains links to tables.
2. On the Tools menu, point to Add-ins (Database Utilities in Access

2000 or newer), and then click `Linked Table Manager`.

3. Select the `Always Prompt For A New Location` check box.
4. Select the check box for the tables whose links you want to change, and then click `OK`.
5. In the `Select New Location of <table name>` dialog box, specify the new location, click `Open`, and then click `OK`.

23.1.4.4. MyODBC Programming Examples

With a suitable ODBC Manager and the my MyODBC driver installed, any programming language or environment that can support ODBC should be able to connect to a MySQL database through MyODBC.

This includes, but is certainly not limited to, Microsoft support languages (including Visual Basic, C# and interfaces such as ODBC.NET), Perl (through the DBI module, and the `DBD::ODBC` driver).

23.1.4.4.1. Using MyODBC with Visual Basic Using ADO, DAO and RDO

This section contains simple examples of the use of MySQL ODBC 3.51 Driver with ADO, DAO and RDO.

23.1.4.4.1.1. ADO: `rs.addNew`, `rs.delete`, and `rs.update`

The following ADO (ActiveX Data Objects) example creates a table `my_ado` and demonstrates the use of `rs.addNew`, `rs.delete`, and `rs.update`.

```
Private Sub myodbc_ado_Click()  
  
Dim conn As ADODB.Connection  
Dim rs As ADODB.Recordset  
Dim fld As ADODB.Field  
Dim sql As String  
  
'connect to MySQL server using MySQL ODBC 3.51 Driver  
Set conn = New ADODB.Connection  
conn.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"  
& "SERVER=localhost;"  
& " DATABASE=test;"
```

```
& "UID=venu;PWD=venu; OPTION=3"
```

```
conn.Open
```

```
'create table
```

```
conn.Execute "DROP TABLE IF EXISTS my_ado"
```

```
conn.Execute "CREATE TABLE my_ado(id int not null primary key, name  
& "txt text, dt date, tm time, ts timestamp)"
```

```
'direct insert
```

```
conn.Execute "INSERT INTO my_ado(id,name,txt) values(1,100,'venu')"
```

```
conn.Execute "INSERT INTO my_ado(id,name,txt) values(2,200,'MySQL')"
```

```
conn.Execute "INSERT INTO my_ado(id,name,txt) values(3,300,'Delete')"
```

```
Set rs = New ADODB.Recordset
```

```
rs.CursorLocation = adUseServer
```

```
'fetch the initial table ..
```

```
rs.Open "SELECT * FROM my_ado", conn
```

```
Debug.Print rs.RecordCount
```

```
rs.MoveFirst
```

```
Debug.Print String(50, "-") & "Initial my_ado Result Set " & String(
```

```
For Each fld In rs.Fields
```

```
Debug.Print fld.Name,
```

```
Next
```

```
Debug.Print
```

```
Do Until rs.EOF
```

```
For Each fld In rs.Fields
```

```
Debug.Print fld.Value,
```

```
Next
```

```
rs.MoveNext
```

```
Debug.Print
```

```
Loop
```

```
rs.Close
```

```
'rs insert
```

```
rs.Open "select * from my_ado", conn, adOpenDynamic, adLockOptimisti
```

```
rs.AddNew
```

```
rs!Name = "Monty"
```

```
rs!txt = "Insert row"
```

```
rs.Update
```

```
rs.Close
```

```
'rs update
```

```
rs.Open "SELECT * FROM my_ado"
```

```
rs!Name = "update"
```

```
rs!txt = "updated-row"
```

```
rs.Update
```

```
rs.Close
```

```

'rs update second time..
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-second-time"
rs.Update
rs.Close

'rs delete
rs.Open "SELECT * FROM my_ado"
rs.MoveNext
rs.MoveNext
rs.Delete
rs.Close

'fetch the updated table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Updated my_ado Result Set " & String(
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print

Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close
conn.Close
End Sub

```

23.1.4.4.1.2. DAO: rs.addNew, rs.update, and Scrolling

The following DAO (Data Access Objects) example creates a table my_dao and demonstrates the use of rs.addNew, rs.update, and result set scrolling.

```

Private Sub myodbc_dao_Click()

Dim ws As Workspace
Dim conn As Connection
Dim queryDef As queryDef
Dim str As String

```

```

'connect to MySQL using MySQL ODBC 3.51 Driver
Set ws = DBEngine.CreateWorkspace("", "venu", "venu", dbUseODBC)
str = "odbc;DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"
Set conn = ws.OpenConnection("test", dbDriverNoPrompt, False, str)

'Create table my_dao
Set queryDef = conn.CreateQueryDef("", "drop table if exists my_dao")
queryDef.Execute

Set queryDef = conn.CreateQueryDef("", "create table my_dao(Id INT A
& "Ts TIMESTAMP(14) NOT NULL, Name varchar(20), Id2 INT)")
queryDef.Execute

'Insert new records using rs.addNew
Set rs = conn.OpenRecordset("my_dao")
Dim i As Integer

For i = 10 To 15
rs.AddNew
rs!Name = "insert record" & i
rs!Id2 = i
rs.Update
Next i
rs.Close

'rs update..
Set rs = conn.OpenRecordset("my_dao")
rs.Edit
rs!Name = "updated-string"
rs.Update
rs.Close

'fetch the table back...
Set rs = conn.OpenRecordset("my_dao", dbOpenDynamic)
str = "Results:"
rs.MoveFirst
While Not rs.EOF
str = " " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print "DATA:" & str
rs.MoveNext
Wend

'rs Scrolling
rs.MoveFirst
str = " FIRST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , "
Debug.Print str

```

```

rs.MoveLast
str = " LAST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , "
Debug.Print str

rs.MovePrevious
str = " LAST-1 ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " ,
Debug.Print str

'free all resources
rs.Close
queryDef.Close
conn.Close
ws.Close

End Sub

```

23.1.4.4.1.3. RDO: rs.addNew and rs.update

The following RDO (Remote Data Objects) example creates a table my_rdo and demonstrates the use of rs.addNew and rs.update.

```

Dim rs As rdoResultset
Dim cn As New rdoConnection
Dim cl As rdoColumn
Dim SQL As String

'cn.Connect = "DSN=test;"
cn.Connect = "DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"

cn.CursorDriver = rdUseOdbc
cn.EstablishConnection rdDriverPrompt

'drop table my_rdo
SQL = "drop table if exists my_rdo"
cn.Execute SQL, rdExecDirect

'create table my_rdo
SQL = "create table my_rdo(id int, name varchar(20))"
cn.Execute SQL, rdExecDirect

'insert - direct
SQL = "insert into my_rdo values (100,'venu')"
cn.Execute SQL, rdExecDirect

```

```

SQL = "insert into my_rdo values (200,'MySQL')"
cn.Execute SQL, rdExecDirect

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecD
rs.AddNew
rs!id = 300
rs!Name = "Insert1"
rs.Update
rs.Close

'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecD
rs.AddNew
rs!id = 400
rs!Name = "Insert 2"
rs.Update
rs.Close

'rs update
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecD
rs.Edit
rs!id = 999
rs!Name = "updated"
rs.Update
rs.Close

'fetch back...
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecD
Do Until rs.EOF
For Each cl In rs.rdoColumns
Debug.Print cl.Value,
Next
rs.MoveNext
Debug.Print
Loop
Debug.Print "Row count="; rs.RowCount

'close
rs.Close
cn.Close

End Sub

```

23.1.4.4.2. Using MyODBC with .NET

This section contains simple examples that demonstrate the use of MyODBC drivers with ODBC.NET.

23.1.4.4.2.1. Using MyODBC with ODBC.NET and C# (C sharp)

The following sample creates a table my_odbc_net and demonstrates its use in C#.

```
/**
 * @sample      : mycon.cs
 * @purpose     : Demo sample for ODBC.NET using MyODBC
 * @author      : Venu, <myodbc@lists.mysql.com>
 *
 * (C) Copyright MySQL AB, 1995-2006
 *
 **/

/* build command
 *
 * csc /t:exe
 *      /out:mycon.exe mycon.cs
 *      /r:Microsoft.Data.Odbc.dll
 */

using Console = System.Console;
using Microsoft.Data.Odbc;

namespace myodbc3
{
    class mycon
    {
        static void Main(string[] args)
        {
            try
            {
                //Connection string for MyODBC 2.50
                /*string MyConString = "DRIVER={MySQL};" +
                "SERVER=localhost;" +
                "DATABASE=test;" +
                "UID=venu;" +
                "PASSWORD=venu;" +
                "OPTION=3";
                */
                //Connection string for MyODBC 3.51
                string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
                "SERVER=localhost;" +
                "DATABASE=test;" +
```

```

    "UID=venu;" +
    "PASSWORD=venu;" +
    "OPTION=3";

//Connect to MySQL using MyODBC
OdbcConnection MyConnection = new OdbcConnection(MyConStri
MyConnection.Open());

Console.WriteLine("\n !!! success, connected successfully

//Display connection information
Console.WriteLine("Connection Information:");
Console.WriteLine("\tConnection String:" +
    MyConnection.ConnectionString);
Console.WriteLine("\tConnection Timeout:" +
    MyConnection.ConnectionTimeout);
Console.WriteLine("\tDatabase:" +
    MyConnection.Database);
Console.WriteLine("\tDataSource:" +
    MyConnection.DataSource);
Console.WriteLine("\tDriver:" +
    MyConnection.Driver);
Console.WriteLine("\tServerVersion:" +
    MyConnection.ServerVersion);

//Create a sample table
OdbcCommand MyCommand =
    new OdbcCommand("DROP TABLE IF EXISTS my_odbc_net",
        MyConnection);
MyCommand.ExecuteNonQuery();
MyCommand.CommandText =
    "CREATE TABLE my_odbc_net(id int, name varchar(20), idb
MyCommand.ExecuteNonQuery();

//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(10, 'venu', 300)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());;

//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(20, 'mysql', 400)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());;

//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(20, 'mysql', 500)";

```

```

Console.WriteLine("INSERT, Total rows affected:" +
                  MyCommand.ExecuteNonQuery());

//Update
MyCommand.CommandText =
    "UPDATE my_odbc_net SET id=999 WHERE id=20";
Console.WriteLine("Update, Total rows affected:" +
                  MyCommand.ExecuteNonQuery());

//COUNT(*)
MyCommand.CommandText =
    "SELECT COUNT(*) as TRows FROM my_odbc_net";
Console.WriteLine("Total Rows:" +
                  MyCommand.ExecuteScalar());

//Fetch
MyCommand.CommandText = "SELECT * FROM my_odbc_net";
OdbcDataReader MyDataReader;
MyDataReader = MyCommand.ExecuteReader();
while (MyDataReader.Read())
{
    if(string.Compare(MyConnection.Driver, "myodbc3.dll") =
        //Supported only by MyODBC 3.51
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0)
                          MyDataReader.GetString(1) + " " +
                          MyDataReader.GetInt64(2));
    }
    else {
        //BIGINTs not supported by MyODBC
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0)
                          MyDataReader.GetString(1) + " " +
                          MyDataReader.GetInt32(2));
    }
}

//Close all resources
MyDataReader.Close();
MyConnection.Close();
}
catch (OdbcException MyOdbcException) //Catch any ODBC excepti
{
    for (int i=0; i < MyOdbcException.Errors.Count; i++)
    {
        Console.Write("ERROR #" + i + "\n" +
                      "Message: " +
                      MyOdbcException.Errors[i].Message + "\n"
                      "Native: " +
                      MyOdbcException.Errors[i].NativeError.To
                      "Source: " +

```



```

MyConnection.Open()

Console.WriteLine("Connection State::" & MyConnection.State.To

'Drop
Console.WriteLine("Dropping table")
Dim MyCommand As New OdbcCommand()
MyCommand.Connection = MyConnection
MyCommand.CommandText = "DROP TABLE IF EXISTS my_vb_net"
MyCommand.ExecuteNonQuery()

'Create
Console.WriteLine("Creating...")
MyCommand.CommandText = "CREATE TABLE my_vb_net(id int, name v
MyCommand.ExecuteNonQuery()

'Insert
MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(10,'venu
Console.WriteLine("INSERT, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'Insert
MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysq
Console.WriteLine("INSERT, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'Insert
MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysq
Console.WriteLine("INSERT, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'Insert
MyCommand.CommandText = "INSERT INTO my_vb_net(id) VALUES(30)"
Console.WriteLine("INSERT, Total rows affected:" & _
                MyCommand.ExecuteNonQuery()

'Update
MyCommand.CommandText = "UPDATE my_vb_net SET id=999 WHERE id=
Console.WriteLine("Update, Total rows affected:" & _
MyCommand.ExecuteNonQuery()

'COUNT(*)
MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_vb_n
Console.WriteLine("Total Rows:" & MyCommand.ExecuteScalar())

'Select
Console.WriteLine("Select * FROM my_vb_net")
MyCommand.CommandText = "SELECT * FROM my_vb_net"
Dim MyDataReader As OdbcDataReader

```

```

MyDataReader = MyCommand.ExecuteReader
While MyDataReader.Read
    If MyDataReader("name") Is DBNull.Value Then
        Console.WriteLine("id = " & _
            CStr(MyDataReader("id")) & " name = " & _
            "NULL")
    Else
        Console.WriteLine("id = " & _
            CStr(MyDataReader("id")) & " name = " & _
            CStr(MyDataReader("name")))
    End If
End While

'Catch ODBC Exception
Catch MyOdbcException As OdbcException
    Dim i As Integer
    Console.WriteLine(MyOdbcException.ToString)

'Catch program exception
Catch MyException As Exception
    Console.WriteLine(MyException.ToString)
End Try
End Sub

```

23.1.5. MyODBC Reference

This section provides reference material for the MyODBC API, showing supported functions and methods, supported MySQL column types and the corresponding native type in MyODBC, and the error codes returned by MyODBC when a fault occurs.

23.1.5.1. MyODBC API Reference

This section summarizes ODBC routines, categorized by functionality.

For the complete ODBC API reference, please refer to the ODBC Programmer's Reference at http://msdn.microsoft.com/library/en-us/odbc/html/odbcabout_this_manual.asp.

An application can call `SQLGetInfo` function to obtain conformance information about MyODBC. To obtain information about support for a specific function in the driver, an application can call `SQLGetFunctions`.

Note: For backward compatibility, the MyODBC 3.51 driver supports all

deprecated functions.

The following tables list MyODBC API calls grouped by task:

Connecting to a data source:

	MyODBC			
Function name	2.50	3.51	Standard	Purpose
SQLAllocHandle	No	Yes	ISO 92	Obtains an environment, connection, statement, or descriptor handle.
SQLConnect	Yes	Yes	ISO 92	Connects to a specific driver by data source name, user ID, and password.
SQLDriverConnect	Yes	Yes	ODBC	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
SQLAllocEnv	Yes	Yes	Deprecated	Obtains an environment handle allocated from driver.
SQLAllocConnect	Yes	Yes	Deprecated	Obtains a connection handle

Obtaining information about a driver and data source:

	MyODBC			
Function name	2.50	3.51	Standard	Purpose
SQLDataSources	No	No	ISO 92	Returns the list of available data sources, handled by the Driver Manager
SQLDrivers	No	No	ODBC	Returns the list of installed drivers and their attributes, handles by Driver Manager
SQLGetInfo	Yes	Yes	ISO 92	Returns information about a specific driver and data source.
SQLGetFunctions	Yes	Yes	ISO 92	Returns supported driver functions.

SQLGetTypeInfo	Yes	Yes	ISO 92	Returns information about supported data types.
----------------	-----	-----	--------	---

Setting and retrieving driver attributes:

Function name	MyODBC		Standard	Purpose
	2.50	3.51		
SQLSetConnectAttr	No	Yes	ISO 92	Sets a connection attribute.
SQLGetConnectAttr	No	Yes	ISO 92	Returns the value of a connection attribute.
SQLSetConnectOption	Yes	Yes	Deprecated	Sets a connection option
SQLGetConnectOption	Yes	Yes	Deprecated	Returns the value of a connection option
SQLSetEnvAttr	No	Yes	ISO 92	Sets an environment attribute.
SQLGetEnvAttr	No	Yes	ISO 92	Returns the value of an environment attribute.
SQLSetStmtAttr	No	Yes	ISO 92	Sets a statement attribute.
SQLGetStmtAttr	No	Yes	ISO 92	Returns the value of a statement attribute.
SQLSetStmtOption	Yes	Yes	Deprecated	Sets a statement option
SQLGetStmtOption	Yes	Yes	Deprecated	Returns the value of a statement option

Preparing SQL requests:

Function name	MyODBC		Standard	Purpose
	2.50	3.51		
SQLAllocStmt	Yes	Yes	Deprecated	Allocates a statement handle
SQLPrepare	Yes	Yes	ISO 92	Prepares an SQL statement for later execution.
SQLBindParameter	Yes	Yes	ODBC	Assigns storage for a parameter in an SQL statement.
SQLGetCursorName	Yes	Yes	ISO 92	Returns the cursor name associated with a statement

				handle.
SQLSetCursorName	Yes	Yes	ISO 92	Specifies a cursor name.
SQLSetScrollOptions	Yes	Yes	ODBC	Sets options that control cursor behavior.

Submitting requests:

	MyODBC			
Function name	2.50	3.51	Standard	Purpose
SQLExecute	Yes	Yes	ISO 92	Executes a prepared statement.
SQLExecDirect	Yes	Yes	ISO 92	Executes a statement
SQLNativeSql	Yes	Yes	ODBC	Returns the text of an SQL statement as translated by the driver.
SQLDescribeParam	Yes	Yes	ODBC	Returns the description for a specific parameter in a statement.
SQLNumParams	Yes	Yes	ISO 92	Returns the number of parameters in a statement.
SQLParamData	Yes	Yes	ISO 92	Used in conjunction with SQLPutData to supply parameter data at execution time. (Useful for long data values.)
SQLPutData	Yes	Yes	ISO 92	Sends part or all of a data value for a parameter. (Useful for long data values.)

Retrieving results and information about results:

	MyODBC			
Function name	2.50	3.51	Standard	Purpose
SQLRowCount	Yes	Yes	ISO 92	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResultCols	Yes	Yes	ISO 92	Returns the number of columns in the result set.
				Describes a column in the result

SQLDescribeCol	Yes	Yes	ISO 92	set.
SQLColAttribute	No	Yes	ISO 92	Describes attributes of a column in the result set.
SQLColAttributes	Yes	Yes	Deprecated	Describes attributes of a column in the result set.
SQLFetch	Yes	Yes	ISO 92	Returns multiple result rows.
SQLFetchScroll	No	Yes	ISO 92	Returns scrollable result rows.
SQLExtendedFetch	Yes	Yes	Deprecated	Returns scrollable result rows.
SQLSetPos	Yes	Yes	ODBC	Positions a cursor within a fetched block of data and allows an application to refresh data in the rowset or to update or delete data in the result set.
SQLBulkOperations	No	Yes	ODBC	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.

Retrieving error or diagnostic information:

	MyODBC			
Function name	2.50	3.51	Standard	Purpose
SQLError	Yes	Yes	Deprecated	Returns additional error or status information
SQLGetDiagField	Yes	Yes	ISO 92	Returns additional diagnostic information (a single field of the diagnostic data structure).
SQLGetDiagRec	Yes	Yes	ISO 92	Returns additional diagnostic information (multiple fields of the diagnostic data structure).

Obtaining information about the data source's system tables (catalog functions) item:

	MyODBC		

Function name	2.50	3.51	Standard	Purpose
SQLColumnPrivileges	Yes	Yes	ODBC	Returns a list of columns and associated privileges for one or more tables.
SQLColumns	Yes	Yes	X/Open	Returns the list of column names in specified tables.
SQLForeignKeys	Yes	Yes	ODBC	Returns a list of column names that make up foreign keys, if they exist for a specified table.
SQLPrimaryKeys	Yes	Yes	ODBC	Returns the list of column names that make up the primary key for a table.
SQLSpecialColumns	Yes	Yes	X/Open	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	Yes	Yes	ISO 92	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges	Yes	Yes	ODBC	Returns a list of tables and the privileges associated with each table.
SQLTables	Yes	Yes	X/Open	Returns the list of table names stored in a specific data source.

Performing transactions:

	MyODBC			
Function name	2.50	3.51	Standard	Purpose
SQLTransact	Yes	Yes	Deprecated	Commits or rolls back a transaction
SQLEndTran	No	Yes	ISO 92	Commits or rolls back a transaction.

Terminating a statement:

	MyODBC			
Function name	2.50	3.51	Standard	Purpose
SQLFreeStmt	Yes	Yes	ISO 92	Ends statement processing, discards pending results, and, optionally, frees all resources associated with the statement handle.
SQLCloseCursor	Yes	Yes	ISO 92	Closes a cursor that has been opened on a statement handle.
SQLCancel	Yes	Yes	ISO 92	Cancels an SQL statement.

Terminating a connection:

	MyODBC			
Function name	2.50	3.51	Standard	Purpose
SQLDisconnect	Yes	Yes	ISO 92	Closes the connection.
SQLFreeHandle	No	Yes	ISO 92	Releases an environment, connection, statement, or descriptor handle.
SQLFreeConnect	Yes	Yes	Deprecated	Releases connection handle
SQLFreeEnv	Yes	Yes	Deprecated	Releases an environment handle

23.1.5.2. MyODBC Data Types

The following table illustrates how driver maps the server data types to default SQL and C data types:

Native Value	SQL Type	C Type
bit	SQL_BIT	SQL_C_BIT
tinyint	SQL_TINYINT	SQL_C_STINYINT
tinyint unsigned	SQL_TINYINT	SQL_C_UTINYINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
blob	SQL_LONGVARBINARY	SQL_C_BINARY
longblob	SQL_LONGVARBINARY	SQL_C_BINARY

tinyblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR
mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_CHAR
integer	SQL_INTEGER	SQL_C_SLONG
integer unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
int unsigned	SQL_INTEGER	SQL_C_ULONG
mediumint	SQL_INTEGER	SQL_C_SLONG
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
smallint	SQL_SMALLINT	SQL_C_SSHORT
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT
real	SQL_FLOAT	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
float	SQL_REAL	SQL_C_FLOAT
double precision	SQL_DOUBLE	SQL_C_DOUBLE
date	SQL_DATE	SQL_C_DATE
time	SQL_TIME	SQL_C_TIME
year	SQL_SMALLINT	SQL_C_SHORT
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
timestamp	SQL_TIMESTAMP	SQL_C_TIMESTAMP
text	SQL_VARCHAR	SQL_C_CHAR
varchar	SQL_VARCHAR	SQL_C_CHAR
enum	SQL_VARCHAR	SQL_C_CHAR
set	SQL_VARCHAR	SQL_C_CHAR
bit	SQL_CHAR	SQL_C_CHAR
bool	SQL_CHAR	SQL_C_CHAR

23.1.5.3. MyODBC Error Codes

The following tables lists the error codes returned by the driver apart from the server errors.

--	--	--	--

Native Code	SQLSTATE 2	SQLSTATE 3	Error Message
500	01000	01000	General warning
501	01004	01004	String data, right truncated
502	01S02	01S02	Option value changed
503	01S03	01S03	No rows updated/deleted
504	01S04	01S04	More than one row updated/deleted
505	01S06	01S06	Attempt to fetch before the result set returned the first row set
506	07001	07002	SQLBindParameter not used for all parameters
507	07005	07005	Prepared statement not a cursor-specification
508	07009	07009	Invalid descriptor index
509	08002	08002	Connection name in use
510	08003	08003	Connection does not exist
511	24000	24000	Invalid cursor state
512	25000	25000	Invalid transaction state
513	25S01	25S01	Transaction state unknown
514	34000	34000	Invalid cursor name
515	S1000	HY000	General driver defined error
516	S1001	HY001	Memory allocation error
517	S1002	HY002	Invalid column number
518	S1003	HY003	Invalid application buffer type
519	S1004	HY004	Invalid SQL data type
520	S1009	HY009	Invalid use of null pointer
521	S1010	HY010	Function sequence error
522	S1011	HY011	Attribute can not be set now
523	S1012	HY012	Invalid transaction operation code
524	S1013	HY013	Memory management error
525	S1015	HY015	No cursor name available
526	S1024	HY024	Invalid attribute value

527	S1090	HY090	Invalid string or buffer length
528	S1091	HY091	Invalid descriptor field identifier
529	S1092	HY092	Invalid attribute/option identifier
530	S1093	HY093	Invalid parameter number
531	S1095	HY095	Function type out of range
532	S1106	HY106	Fetch type out of range
533	S1117	HY117	Row value out of range
534	S1109	HY109	Invalid cursor position
535	S1C00	HYC00	Optional feature not implemented
0	21S01	21S01	Column count does not match value count
0	23000	23000	Integrity constraint violation
0	42000	42000	Syntax error or access violation
0	42S02	42S02	Base table or view not found
0	42S12	42S12	Index not found
0	42S21	42S21	Column already exists
0	42S22	42S22	Column not found
0	08S01	08S01	Communication link failure

23.1.6. MyODBC Notes and Tips

Here are some common notes and tips for using MyODBC within different environments, applications and tools. The notes provided here are based on the experiences of MyODBC developers and users.

23.1.6.1. MyODBC General Functionality

This section provides help with common queries and areas of functionality in MySQL and how to use them with MyODBC.

23.1.6.1.1. Obtaining Auto-Increment Values

Obtaining the value of column that uses `AUTO_INCREMENT` after an `INSERT` statement can be achieved in a number of different ways. To obtain the value

immediately after an INSERT, use a SELECT query with the LAST_INSERT_ID() function.

For example, using MyODBC you would execute two separate statements, the INSERT statement and the SELECT query to obtain the auto-increment value.

```
INSERT INTO tbl (auto,text) VALUES(NULL, 'text');  
SELECT LAST_INSERT_ID();
```

If you do not require the value within your application, but do require the value as part of another INSERT, the entire process can be handled by executing the following statements:

```
INSERT INTO tbl (auto,text) VALUES(NULL, 'text');  
INSERT INTO tbl2 (id,text) VALUES(LAST_INSERT_ID(), 'text');
```

Certain ODBC applications (including Delphi and Access) may have trouble obtaining the auto-increment value using the previous examples. In this case, try the following statement as an alternative:

```
SELECT * FROM tbl WHERE auto IS NULL;
```

See [Section 22.2.13.3, “How to Get the Unique ID for the Last Inserted Row”](#).

23.1.6.1.2. Dynamic Cursor Support

Support for the dynamic cursor is provided in MyODBC 3.51, but dynamic cursors are not enabled by default. You can enable this function within Windows by selecting the Enable Dynamic Cursor checkbox within the ODBC Data Source Administrator.

On other platforms, you can enable the dynamic cursor by adding 32 to the OPTION value when creating the DSN.

23.1.6.1.3. MyODBC Performance

The MyODBC driver has been optimized to provide very fast performance. If you experience problems with the performance of MyODBC, or notice a large amount of disk activity for simple queries, there are a number of aspects you should check:

- Ensure that ODBC Tracing is not enabled. With tracing enabled, a lot of information is recorded in the tracing file by the ODBC Manager. You can check, and disable, tracing within Windows using the Tracing panel of the ODBC Data Source Administrator. Within Mac OS X, check the Tracing panel of ODBC Administrator. See [Section 23.1.3.8, “Getting an ODBC Trace File”](#).
- Make sure you are using the standard version of the driver, and not the debug version. The debug version includes additional checks and reporting measures.
- Disable the MyODBC driver trace and query logs. These options are enabled for each DSN, so make sure to examine only the DSN that you are using in your application. Within Windows, you can disable the MyODBC and query logs by modifying the DSN configuration. Within Mac OS X and Unix, ensure that the driver trace (option value 4) and query logging (option value 524288) are not enabled.

23.1.6.1.4. Setting ODBC Query Timeout in Windows

For more information on how to set the query timeout on Microsoft Windows when executing queries through an ODBC connection, read the Microsoft knowledgebase document at <http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B153756>.

23.1.6.2. MyODBC Application Specific Tips

Most programs should work with MyODBC, but for each of those listed here, there are specific notes and tips to improve or enhance the way you work with MyODBC and these applications.

With all applications you should ensure that you are using the latest MyODBC drivers, ODBC Manager and any supporting libraries and interfaces used by your application. For example, on Windows, using the latest version of Microsoft Data Access Components (MDAC) will improve the compatibility with ODBC in general, and with the MyODBC driver.

23.1.6.2.1. Using MyODBC with Microsoft Applications

The majority of Microsoft applications have been tested with MyODBC, including Microsoft Office, Microsoft Access and the various programming languages supported within ASP and Microsoft Visual Studio.

If you have problem with MyODBC and your program also works with OLEDB, you should try the OLEDB driver.

23.1.6.2.1.1. Microsoft Access

To improve the integration between Microsoft Access and MySQL through MyODBC:

- For all versions of Access, you should enable the MyODBC Return matching rows option. For Access 2.0, you should additionally enable the Simulate ODBC 1.0 option.
- You should have a `TIMESTAMP` column in all tables that you want to be able to update. For maximum portability, don't use a length specification in the column declaration (which is unsupported within MySQL in versions earlier than 4.1).
- You should have a primary key in each MySQL table you want to use with Access. If not, new or updated rows may show up as `#DELETED#`.
- Use only `DOUBLE` float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as `#DELETED#` or that you can't find or update rows.
- If you are using MyODBC to link to a table that has a `BIGINT` column, the results are displayed as `#DELETED#`. The work around solution is:
 - Have one more dummy column with `TIMESTAMP` as the data type.
 - Select the Change `BIGINT` columns to `INT` option in the connection dialog in ODBC DSN Administrator.
 - Delete the table link from Access and re-create it.

Old records may still display as `#DELETED#`, but newly added/updated records are displayed properly.

- If you still get the error Another user has changed your data after adding a `TIMESTAMP` column, the following trick may help you:

Don't use a table data sheet view. Instead, create a form with the fields you want, and use that form data sheet view. You should set the `DefaultValue` property for the `TIMESTAMP` column to `NOW()`. It may be a good idea to hide the `TIMESTAMP` column from view so your users are not confused.

- In some cases, Access may generate SQL statements that MySQL can't understand. You can fix this by selecting "Query|SQLSpecific|Pass-Through" from the Access menu.
- On Windows NT, Access reports `BLOB` columns as `OLE OBJECTS`. If you want to have `MEMO` columns instead, you should change `BLOB` columns to `TEXT` with `ALTER TABLE`.
- Access can't always handle the MySQL `DATE` column properly. If you have a problem with these, change the columns to `DATETIME`.
- If you have in Access a column defined as `BYTE`, Access tries to export this as `TINYINT` instead of `TINYINT UNSIGNED`. This gives you problems if you have values larger than 127 in the column.
- If you have very large (long) tables in Access, it might take a very long time to open them. Or you might run low on virtual memory and eventually get an `ODBC Query Failed` error and the table cannot open. To deal with this, select the following options:
 - Return Matching Rows (2)
 - Allow BIG Results (8).

These add up to a value of 10 (`OPTION=10`).

Some external articles and tips that may be useful when using Access, ODBC and MyODBC:

- Read [How to Trap ODBC Login Error Messages in Access](#)
- Optimizing Access ODBC Applications

- [Optimizing for Client/Server Performance](#)
 - [Tips for Converting Applications to Using ODBC Direct](#)
 - [Tips for Optimizing Queries on Attached SQL Tables](#)
- For a list of tools that can be used with Access and ODBC data sources, refer to [converters](#) section for list of available tools.

23.1.6.2.1.2. Microsoft Excel and Column Types

If you have problems importing data into Microsoft Excel, particularly numerical, date, and time values, this is probably because of a bug in Excel, where the column type of the source data is used to determine the data type when that data is inserted into a cell within the worksheet. The result is that Excel incorrectly identifies the content and this affects both the display format and the data when it is used within calculations.

To address this issue, use the `CONCAT()` function in your queries. The use of `CONCAT()` forces Excel to treat the value as a string, which Excel will then parse and usually correctly identify the embedded information.

However, even with this option, some data may be incorrectly formatted, even though the source data remains unchanged. Use the `Format Cells` option within Excel to change the format of the displayed information.

23.1.6.2.1.3. Microsoft Visual Basic

To be able to update a table, you must define a primary key for the table.

Visual Basic with ADO can't handle big integers. This means that some queries like `SHOW PROCESSLIST` do not work properly. The fix is to use `OPTION=16384` in the ODBC connect string or to select the `Change BIGINT columns to INT` option in the MyODBC connect screen. You may also want to select the `Return matching rows` option.

23.1.6.2.1.4. Microsoft Visual InterDev

If you have a `BIGINT` in your result, you may get the error `[Microsoft][ODBC`

Driver Manager] Driver does not support this parameter. Try selecting the Change BIGINT columns to INT option in the MyODBC connect screen.

23.1.6.2.1.5. Visual Objects

You should select the Don't optimize column widths option.

23.1.6.2.1.6. Microsoft ADO

When you are coding with the ADO API and MyODBC, you need to pay attention to some default properties that aren't supported by the MySQL server. For example, using the CursorLocation Property as adUseServer returns a result of -1 for the RecordCount Property. To have the right value, you need to set this property to adUseClient, as shown in the VB code here:

```
Dim myconn As New ADODB.Connection
Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long

myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount

myrs.Close
myconn.Close
```

Another workaround is to use a SELECT COUNT(*) statement for a similar query to get the correct row count.

To find the number of rows affected by a specific SQL statement in ADO, use the RecordsAffected property in the ADO execute method. For more information on the usage of execute method, refer to <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/htm/mdmthcnexecute.asp>.

For information, see [ActiveX Data Objects\(ADO\) Frequently Asked Questions](#).

23.1.6.2.1.7. Using MyODBC with Active Server Pages (ASP)

You should select the Return matching rows option in the DSN.

For more information about how to access MySQL via ASP using MyODBC, refer to the following articles:

- [Using MyODBC To Access Your MySQL Database Via ASP](#)
- [ASP and MySQL at DWAM.NT](#)

A Frequently Asked Questions list for ASP can be found at <http://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp>.

23.1.6.2.1.8. Using MyODBC with Visual Basic (ADO, DAO and RDO) and ASP

Some articles that may help with Visual Basic and ASP:

- [MySQL BLOB columns and Visual Basic 6](#) by Mike Hillyer (<mike@openwin.org>).
- [How to map Visual basic data type to MySQL types](#) by Mike Hillyer (<mike@openwin.org>).

23.1.6.2.2. Using MyODBC with Borland Applications

With all Borland applications where the Borland Database Engine (BDE) is used, follow these steps to improve compatibility:

- Update to BDE 3.2 or newer.
- Enable the Don't optimize column widths option in the DSN.
- Enabled the Return matching rows option in the DSN.

23.1.6.2.2.1. Using MyODBC with Borland Builder 4

When you start a query, you can use the Active property or the Open method. Note that Active starts by automatically issuing a SELECT * FROM ... query. That may not be a good thing if your tables are large.

23.1.6.2.2.2. Using MyODBC with Delphi

Also, here is some potentially useful Delphi code that sets up both an ODBC entry and a BDE entry for MyODBC. The BDE entry requires a BDE Alias Editor that is free at a Delphi Super Page near you. (Thanks to Bryan Brunton <bryan@flesherfab.com> for this):

```
fReg:= TRegistry.Create;
fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
fReg.WriteString('Database', 'Documents');
fReg.WriteString('Description', ' ');
fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
fReg.WriteString('Flag', '1');
fReg.WriteString('Password', '');
fReg.WriteString('Port', ' ');
fReg.WriteString('Server', 'xmark');
fReg.WriteString('User', 'winuser');
fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
fReg.WriteString('DocumentsFab', 'MySQL');
fReg.CloseKey;
fReg.Free;
```

```
Memo1.Lines.Add('DATABASE NAME=');
Memo1.Lines.Add('USER NAME=');
Memo1.Lines.Add('ODBC DSN=DocumentsFab');
Memo1.Lines.Add('OPEN MODE=READ/WRITE');
Memo1.Lines.Add('BATCH COUNT=200');
Memo1.Lines.Add('LANGDRIVER=');
Memo1.Lines.Add('MAX ROWS=-1');
Memo1.Lines.Add('SCHEMA CACHE DIR=');
Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');
```

```
AliasEditor.Add('DocumentsFab', 'MySQL', Memo1.Lines);
```

23.1.6.2.2.3. Using MyODBC with C++ Builder

Tested with BDE 3.0. The only known problem is that when the table schema changes, query fields are not updated. BDE, however, does not seem to

recognize primary keys, only the index named PRIMARY, although this has not been a problem.

23.1.6.2.3. Using MyODBC with ColdFusion

The following information is taken from the ColdFusion documentation:

Use the following information to configure ColdFusion Server for Linux to use the unixODBC driver with MyODBC for MySQL data sources. Allaire has verified that MyODBC 2.50.26 works with MySQL 3.22.27 and ColdFusion for Linux. (Any newer version should also work.) You can download MyODBC at <http://dev.mysql.com/downloads/connector/odbc/>.

ColdFusion version 4.5.1 allows you to use the ColdFusion Administrator to add the MySQL data source. However, the driver is not included with ColdFusion version 4.5.1. Before the MySQL driver appears in the ODBC data sources drop-down list, you must build and copy the MyODBC driver to `/opt/coldfusion/lib/libmyodbc.so`.

The Contrib directory contains the program `mysdn-xxx.zip` which allows you to build and remove the DSN registry file for the MyODBC driver on ColdFusion applications.

For more information and guides on using ColdFusion and MyODBC, see the following external sites:

- Refer to [MySQL ColdFusion unixODBC MyODBC and Solaris - how to succeed](#)
- ColdFusion (on Solaris and NT with service pack 5), [How-to: MySQL and ColdFusion](#).
- [Troubleshooting Data Sources and Database Connectivity for Unix Platforms](#).

23.1.6.2.4. Using MyODBC with OpenOffice

Open Office (<http://www.openoffice.org>) [How-to: MySQL + OpenOffice](#). [How-to: OpenOffice + MyODBC + unixODBC](#).

23.1.6.2.5. Using MyODBC with Sambar Server

Sambar Server (<http://www.sambarserver.info>) [How-to: MyODBC + SambarServer + MySQL](#).

23.1.6.2.6. Using MyODBC with Pervasive Software DataJunction

You have to change it to output VARCHAR rather than ENUM, as it exports the latter in a manner that causes MySQL problems.

23.1.6.2.7. Using MyODBC with SunSystems Vision

You should select the Return matching rows option.

23.1.6.3. MyODBC Errors and Resolutions

The following section details some common errors and their suggested fix or alternative solution. If you are still experiencing problems, use the MyODBC mailing list; see [Section 23.1.7.1, “MyODBC Community Support”](#).

Many problems can be resolved by upgrading your MyODBC drivers to the latest available release. On Windows, you should also make sure that you have the latest versions of the Microsoft Data Access Components (MDAC) installed.

24.1.6.3.1:

Question:

Are MyODBC 2.50 applications compatible with MyODBC 3.51?

Answer:

Applications based on MyODBC 2.50 should work fine with MyODBC 3.51 and later versions. If you find something is not working with the latest version of MyODBC which previously worked under an earlier version, please file a bug report. See [Section 23.1.7.2, “How to Report MyODBC Problems or Bugs”](#).

24.1.6.3.2:

Question:

I have installed MyODBC on Windows XP x64 Edition or Windows Server 2003 R2 x64. The installation completed successfully, but the MyODBC driver does not appear in ODBC Data Source Administrator.

Answer:

This is not a bug, but is related to the way Windows x64 editions operate with the ODBC driver. On Windows x64 editions, the MyODBC driver is installed in the %SystemRoot%\SysWOW64 folder. However, the default ODBC Data Source Administrator that is available through the Administrative Tools or Control Panel in Windows x64 Editions is located in the %SystemRoot%\system32 folder, and only searches this folder for ODBC drivers.

On Windows x64 editions, you should use the ODBC administration tool located at %SystemRoot%\SysWOW64\odbcad32.exe, this will correctly locate the installed MyODBC drivers and enable you to create a MyODBC DSN.

This issue was originally reported as Bug #20301.

24.1.6.3.3:

Question:

When connecting or using the Test button in ODBC Data Source Administrator I get error 10061 (Cannot connect to server)

Answer:

This error can be raised by a number of different issues, including server problems, network problems, and firewall and port blocking problems. For more information, see [Section A.2.2, "Can't connect to \[local\] MySQL server"](#).

24.1.6.3.4:

Question:

The following error is reported when using transactions: Transactions are not enabled

Answer:

This error indicates that you are trying to use transactions with a MySQL table that does not support transactions. Transactions are supported within MySQL when using the InnoDB and BDB database engines.

You should check the following before continuing:

- Verify that your MySQL server supports a transactional database engine. Use `SHOW ENGINES` to obtain a list of the available engine types.
- Verify that the tables you are updating use a transaction database engine.
- Ensure that you have not enabled the `disable_transactions` option in your DSN.

24.1.6.3.5:

Question:

The following error is reported when I submit a query: `Cursor not found`

Answer:

This occurs because the application is using the old MyODBC 2.50 version, and it did not set the cursor name explicitly through `SQLSetCursorName`. The fix is to upgrade to MyODBC 3.51 version.

24.1.6.3.6:

Question:

Access reports records as `#DELETED#` when inserting or updating records in linked tables.

Answer:

If the inserted or updated records are shown as `#DELETED#` in the access, then:

- If you are using Access 2000, you should get and install the newest (version 2.6 or higher) Microsoft MDAC (Microsoft Data Access Components) from <http://www.microsoft.com/data/>. This fixes a bug in Access that when you export data to MySQL, the table and column names aren't specified.

Another way to work around this bug is to upgrade to MyODBC 2.50.33 or higher and MySQL 3.23.x or higher, which together provide a workaround for the problem.

You should also get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5) which can be found at <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114>. This fixes some cases where columns are marked as #DELETED# in Access.

Note: If you are using MySQL 3.22, you must apply the MDAC patch and use MyODBC 2.50.32 or 2.50.34 and up to work around this problem.

- For all versions of Access, you should enable the MyODBC Return matching rows option. For Access 2.0, you should additionally enable the Simulate ODBC 1.0 option.
- You should have a timestamp in all tables that you want to be able to update..
- You should have a primary key in the table. If not, new or updated rows may show up as #DELETED#.
- Use only DOUBLE float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as #DELETED# or that you can't find or update rows.
- If you are using MyODBC to link to a table that has a BIGINT column, the results are displayed as #DELETED. The work around solution is:
 - Have one more dummy column with TIMESTAMP as the data type.
 - Select the Change BIGINT columns to INT option in the connection dialog in ODBC DSN Administrator.
 - Delete the table link from Access and re-create it.

Old records still display as #DELETED#, but newly added/updated records are displayed properly.

24.1.6.3.7:

Question:

How do I handle Write Conflicts or Row Location errors?

Answer:

If you see the following errors, select the Return Matching Rows option in the DSN configuration dialog, or specify `OPTION=2`, as the connection parameter:

Write Conflict. Another user has changed your data.

Row cannot be located for updating. Some values may have been change since it was last read.

24.1.6.3.8:

Question:

Exporting data from Access 97 to MySQL reports a Syntax Error.

Answer:

This error is specific to Access 97 and versions of MyODBC earlier than 3.51.02. Update to the latest version of the MyODBC driver to resolve this problem.

24.1.6.3.9:

Question:

Exporting data from Microsoft DTS to MySQL reports a Syntax Error.

Answer:

This error occurs only with MySQL tables using the TEXT or VARCHAR data types. You can fix this error by upgrading your MyODBC driver to version 3.51.02 or higher.

24.1.6.3.10:

Question:

Using ODBC.NET with MyODBC, while fetching empty string (0 length), it starts giving the SQL_NO_DATA exception.

Answer:

You can get the patch that addresses this problem from <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q319243>.

24.1.6.3.11:

Question:

Using `SELECT COUNT(*) FROM tbl_name` within Visual Basic and ASP returns an error.

Answer:

This error occurs because the `COUNT(*)` expression is returning a `BIGINT`, and ADO can't make sense of a number this big. Select the `Change BIGINT columns to INT` option (option value 16384).

24.1.6.3.12:

Question:

Using the `AppendChunk()` or `GetChunk()` ADO methods, the Multiple-step operation generated errors. Check each status value error is returned.

Answer:

The `GetChunk()` and `AppendChunk()` methods from ADO doesn't work as expected when the cursor location is specified as `adUseServer`. On the other hand, you can overcome this error by using `adUseClient`.

A simple example can be found from http://www.dwam.net/iishelp/ado/docs/adomth02_4.htm

24.1.6.3.13:

Question:

Access Returns Another user had modified the record that you have modified while editing records on a Linked Table.

Answer:

In most cases, this can be solved by doing one of the following things:

- Add a primary key for the table if one doesn't exist.
- Add a timestamp column if one doesn't exist.
- Only use double-precision float fields. Some programs may fail when they compare single-precision floats.

If these strategies don't help, you should start by making a log file from the ODBC manager (the log you get when requesting logs from ODBCADMIN) and a MyODBC log to help you figure out why things go wrong. For instructions, see [Section 23.1.3.8, “Getting an ODBC Trace File”](#).

23.1.7. MyODBC Support

There are many different places where you can get support for using MyODBC. You should always try the MyODBC Mailing List or MyODBC Forum. See [Section 23.1.7.1, “MyODBC Community Support”](#), for help before reporting a specific bug or issue to MySQL.

23.1.7.1. MyODBC Community Support

MySQL AB provides assistance to the user community by means of its mailing lists. For MyODBC-related issues, you can get help from experienced users by using the <myodbc@lists.mysql.com> mailing list. Archives are available online at <http://lists.mysql.com/myodbc>.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [Section 1.7.1, “MySQL Mailing Lists”](#).

Community support from experienced users is also available through the [MyODBC Forum](#). You may also find help from other users in the other MySQL

Forums, located at <http://forums.mysql.com>. See [Section 1.7.2, “MySQL Community Support at the MySQL Forums”](#).

23.1.7.2. How to Report MyODBC Problems or Bugs

If you encounter difficulties or problems with MyODBC, you should start by making a log file from the ODBC Manager (the log you get when requesting logs from ODBC ADMIN) and MyODBC. The procedure for doing this is described in [Section 23.1.3.8, “Getting an ODBC Trace File”](#).

Check the MyODBC trace file to find out what could be wrong. You should be able to determine what statements were issued by searching for the string `>mysql_real_query` in the `myodbc.log` file.

You should also try issuing the statements from the **mysql** client program or from `admindemo`. This helps you determine whether the error is in MyODBC or MySQL.

If you find out something is wrong, please only send the relevant rows (maximum 40 rows) to the `myodbc` mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#). Please never send the whole MyODBC or ODBC log file!

You should ideally include the following information with the email:

- Operating system and version
- MyODBC version
- ODBC Driver Manager type and version
- MySQL server version
- ODBC trace from Driver Manager
- MyODBC log file from MyODBC driver
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem!

Also, before posting the bug, check the MyODBC mailing list archive at <http://lists.mysql.com/myodbc>.

If you are unable to find out what's wrong, the last option is to create an archive in **tar** or Zip format that contains a MyODBC trace file, the ODBC log file, and a README file that explains the problem. You can send this to <ftp://ftp.mysql.com/pub/mysql/upload/>. Only MySQL engineers have access to the files you upload, and we are very discreet with the data.

If you can create a program that also demonstrates the problem, please include it in the archive as well.

If the program works with another SQL server, you should include an ODBC log file where you perform exactly the same SQL statements so that we can compare the results between the two systems.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

23.1.7.3. How to Submit a MyODBC Patch

You can send a patch or suggest a better solution for any existing code or problems by sending a mail message to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com).

23.1.7.4. MyODBC Change History

The MyODBC Change History (Changelog) is located with the main Changelog for MySQL. See [Section D.3, “MySQL Connector/ODBC \(MyODBC\) Change History”](#).

23.1.7.5. Credits

These are the developers that have worked on the MyODBC and MyODBC 3.51 Drivers from MySQL AB.

- Michael (Monty) Widenius
- Venu Anuganti

- Peter Harvey

23.2. Connector/NET

Connector/NET enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. Connector/NET is a fully managed ADO.NET driver written in 100% pure C#.

Connector/NET includes full support for:

- MySQL 5.0 features (such as stored procedures)
- MySQL 4.1 features (server-side prepared statements, Unicode, and shared memory access, and so forth)
- Large-packet support for sending and receiving rows and BLOBs up to 2 gigabytes in size.
- Protocol compression which allows for compressing the data stream between the client and server.
- Support for connecting using TCP/IP sockets, named pipes, or shared memory on Windows.
- Support for connecting using TCP/IP sockets or Unix sockets on Unix.
- Support for the Open Source Mono framework developed by Novell.
- Fully managed, does not utilize the MySQL client library.

This document is intended as a user's guide to Connector/NET and includes a full syntax reference. Syntax information is also included within the `Documentation.chm` file included with the Connector/NET distribution.

23.2.1. Connector/NET Versions

There is currently one version of the Connector/NET available:

- Connector/NET 1.0 includes support for MySQL 4.0, and MySQL 5.0 features, and full compatibility with the ADO.NET driver interface.

23.2.2. How to install Connector/NET

Connector/NET runs on any platform that supports the .NET framework. The .NET framework is primarily supported on recent versions of Microsoft Windows, and is supported on Linux through the Open Source Mono framework (see <http://www.mono-project.com>).

Connector/NET is available for download from <http://dev.mysql.com/downloads/connector/net/1.0.html>.

23.2.2.1. Installing Connector/NET on Windows

On Windows, installation is supported either through a binary installation process or by downloading a Zip file with the Connector/NET components.

Before installing, you should ensure that your system is up to date, including installing the latest version of the .NET Framework.

23.2.2.1.1. Installing Connector/NET using the Installer

Using the installer is the most straightforward method of installing Connector/NET on Windows and the installed components include the source code, test code and full reference documentation.

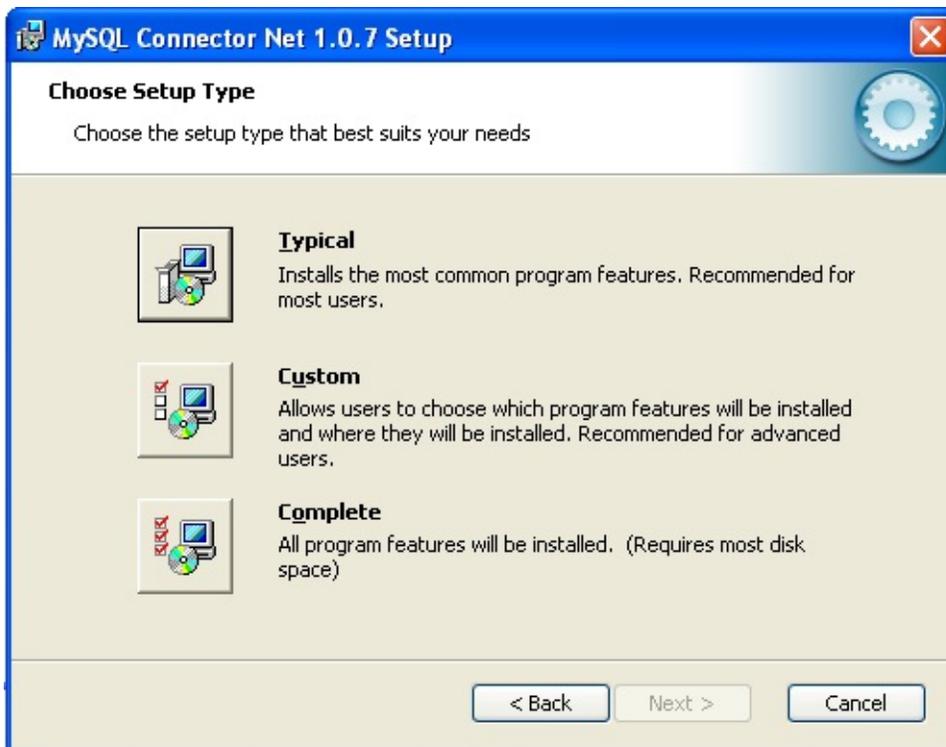
Connector/NET is installed through the use of a Windows Installer (.msi) installation package, which can be used to install Connector/NET on all Windows operating systems. The MSI package is contained within a ZIP archive named `mysql-connector-net-version.zip`, where *version* indicates the Connector/NET version.

To install Connector/NET:

1. Double click on the MSI installer file extracted from the Zip you downloaded. Click Next to start the installation.



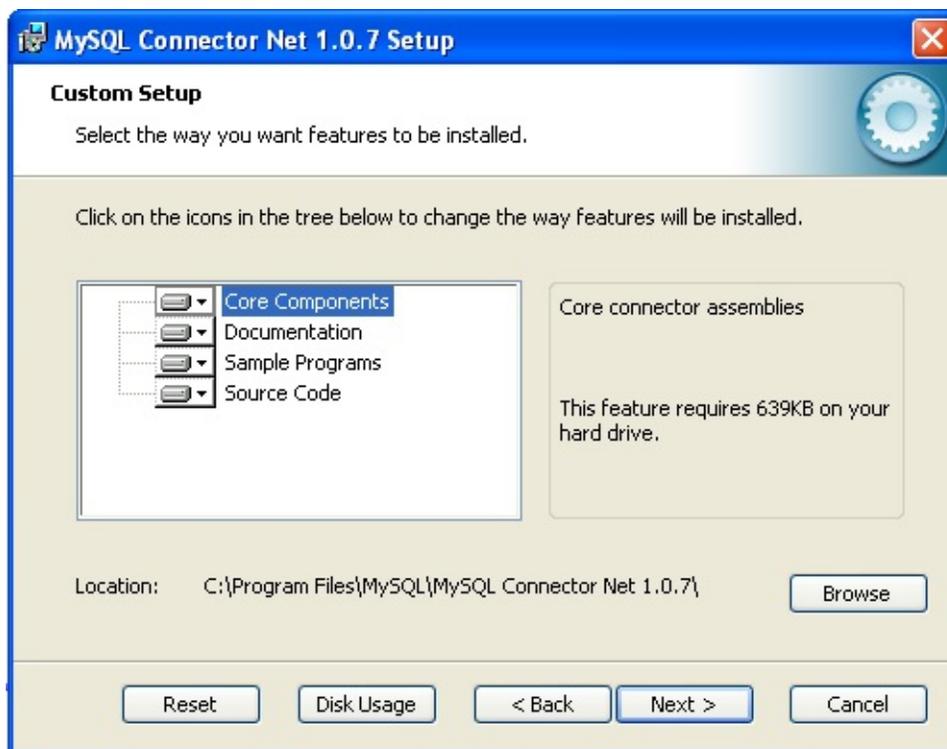
2. You must choose the type of installation that you want to perform.



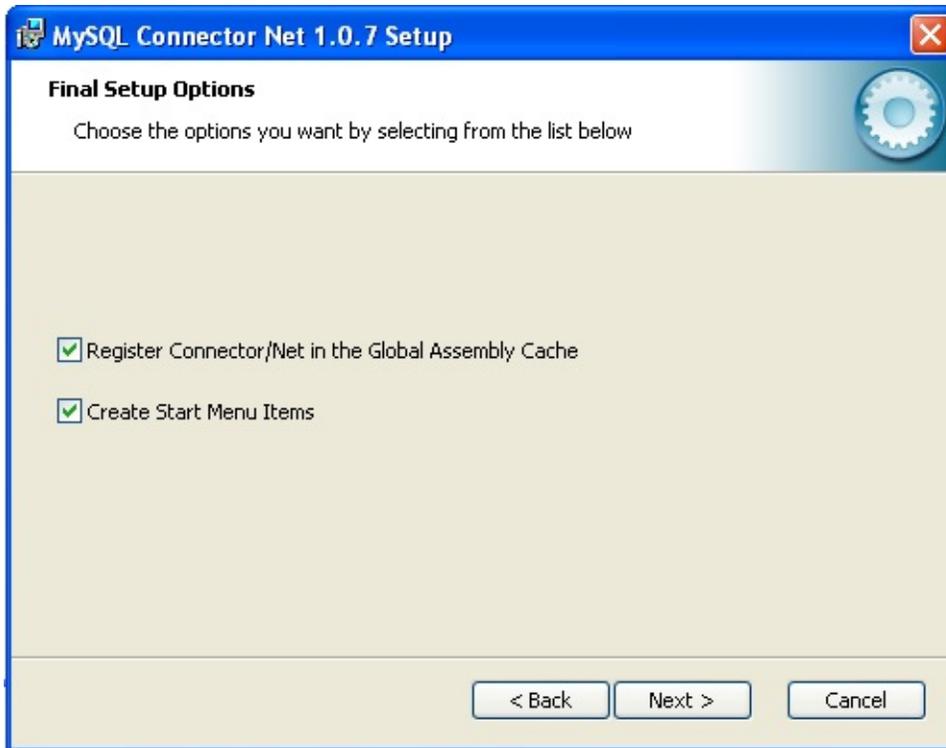
For most situations, the Typical installation will be suitable. Click the

Typical button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the Complete button and proceed to step 5. If you want to customize your installation, including choosing the components to install and some installation options, click the Custom button and proceed to Step 3.

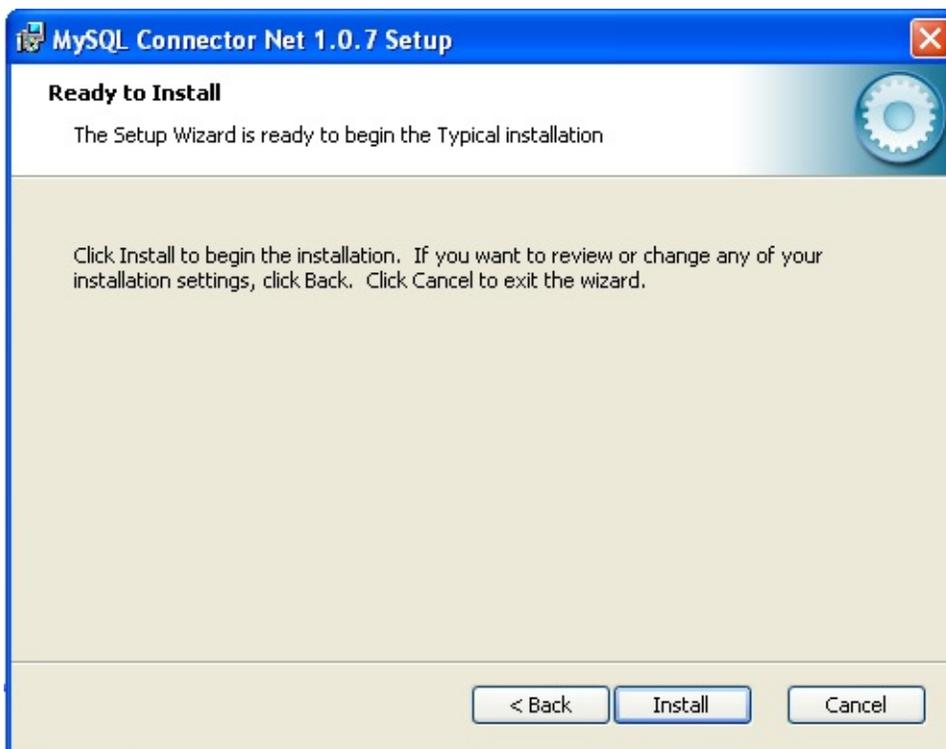
3. If you have chosen a custom installation, you can select the individual components that you want to install, including the core interface component, supporting documentation (a CHM file) samples and examples and the source code. Select the items, and their installation level, and then click Next to continue the installation.



4. For a custom installation you can also decide whether the Connector/NET component should be registered in the Global Assembly Cache - this will make the Connector/NET component available to all applications, not just those where you explicitly reference the Connector/NET component. You can also enable, or disable, the creation or appropriate items in the Start menu. Click Next when you have selected the required options.



5. You will be given a final opportunity to confirm the installation. Click Install to copy and install the files onto your machine.



6. Once the installation has been completed, click Finish to exit the installer.

Unless you choose otherwise, Connector/NET is installed in `C:\Program Files\MySQL\MySQL Connector Net X.X.X`, where `X.X.X` is replaced with the version of Connector/NET you are installing. New installations do not overwrite existing versions of Connector/NET.

Depending on your installation type, the installed components will include some or all of the following components:

- `bin` - Connector/NET MySQL libraries for different versions of the .NET environment.
- `docs` - contains a CHM of the Connector/NET documentation.
- `samples` - sample code and applications that use the Connector/NET component.
- `src` - the source code for the Connector/NET component.

23.2.2.1.2. Installing Connector/NET using the Zip package

If you are having problems running the installer, you can download a .zip file without an installer as an alternative. That file is called `mysql-connector-net-version-noinstall.zip`. Once downloaded, you can extract the files to a location of your choice.

The .zip file contains the following directories:

- `bin` - Connector/NET MySQL libraries for different versions of the .NET environment.
- `doc` - contains a CHM of the Connector/NET documentation.
- `Samples` - sample code and applications that use the Connector/NET component.
- `mysqlclient` - the source code for the Connector/NET component.
- `testsuite` - the test suite used to verify the operation of the

Connector/NET component.

23.2.2.2. Installing Connector/NET on Unix with Mono

There is no installer available for installing the Connector/NET component on your Unix installation. However, the installation is very simple. Before installing, please ensure that you have a working Mono project installation.

Note that you should only install the Connector/NET component on Unix environments where you want to connect to a MySQL server through the Mono project. If you are deploying or developing on a different environment such as Java or Perl then you should use a more appropriate connectivity component. See the [Chapter 23, Connectors](#), or [Chapter 22, APIs and Libraries](#), for more information.

To install Connector/NET on Unix/Mono:

1. Download the `mysql-connector-net-version-noinstall.zip` and extract the contents.
2. Copy the `MySQL.Data.dll` file to your Mono project installation folder.
3. You must register the Connector/NET component in the Global Assembly Cache using the `gacutil` command:

```
shell> gacutil /i MySQL.Data.dll
```

Once installed, applications that are compiled with the Connector/NET component need no further changes. However, you must ensure that when you compile your applications you include the Connector/NET component using the `-r:MySQLData.dll` command line option.

23.2.2.3. Installing Connector/NET using the Source

Caution: You should read this section only if you are interested in helping us test our new code. If you just want to get Connector/NET up and running on your system, you should use a standard release distribution.

To be able to access the Connector/NET source tree, you must have Subversion

installed. Subversion is freely available from <http://subversion.tigris.org/>.

The most recent development source tree is available from our public Subversion trees at <http://dev.mysql.com/tech-resources/sources.html>.

To checkout out the Connector/NET sources, change to the directory where you want the copy of the Connector/NET tree to be stored, then use the following command:

```
shell> svn co  
http://svn.mysql.com/svnpublic/connector-net
```

A Visual Studio project is included in the source which you can use to build Connector/NET.

23.2.3. Connector/NET Examples

Connector/NET comprises several classes that are used to connect to the database, execute queries and statements, and manage query results.

The following are the major classes of Connector/NET:

- **MySQLCommand**: Represents an SQL statement to execute against a MySQL database.
- **MySQLCommandBuilder**: Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated MySQL database.
- **MySQLConnection**: Represents an open connection to a MySQL Server database.
- **MySQLDataAdapter**: Represents a set of data commands and a database connection that are used to fill a dataset and update a MySQL database.
- **MySQLDataReader**: Provides a means of reading a forward-only stream of rows from a MySQL database.
- **MySQLException**: The exception that is thrown when MySQL returns an error.

- `MySQLHelper`: Helper class that makes it easier to work with the provider.
- `MySQLTransaction`: Represents an SQL transaction to be made in a MySQL database.

This section contains basic information and examples for each of the above classes. For a more detailed reference guide please see [Section 23.2.4, “Connector/NET Reference”](#).

23.2.3.1. MySqlCommand

Represents a SQL statement to execute against a MySQL database. This class cannot be inherited.

`MySQLCommand` features the following methods for executing commands at a MySQL database:

Item	Description
ExecuteReader	Executes commands that return rows.
ExecuteNonQuery	Executes commands such as SQL INSERT, DELETE, and UPDATE statements.
ExecuteScalar	Retrieves a single value (for example, an aggregate value) from a database.

You can reset the `CommandText` property and reuse the `MySQLCommand` object. However, you must close the [MySQLDataReader](#) before you can execute a new or previous command.

If a [MySQLException](#) is generated by the method executing a `MySQLCommand`, the [MySQLConnection](#) remains open. It is the responsibility of the programmer to close the connection.

Note. Prior versions of the provider used the '@' symbol to mark parameters in SQL. This is incompatible with MySQL user variables, so the provider now uses the '?' symbol to locate parameters in SQL. To support older code, you can set 'old syntax=yes' on your connection string. If you do this, please be aware that an exception will not be throw if you fail to define a parameter that you intended to use in your SQL.

Examples

The following example creates a [MySqlCommand](#) and a [MySqlConnection](#) . The [MySqlConnection](#) is opened and set as the [Connection](#) for the [MySqlCommand](#) . The example then calls [ExecuteNonQuery](#) , and closes the connection. To accomplish this, the [ExecuteNonQuery](#) is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
Public Sub InsertRow(myConnectionString As String)
    " If the connection string is null, use a default.
    If myConnectionString = "" Then
        myConnectionString = "Database=Test;Data Source=localhost;User I
    End If
    Dim myConnection As New MySqlConnection(myConnectionString)
    Dim myInsertQuery As String = "INSERT INTO Orders (id, customerId,
    Dim myCommand As New MySqlCommand(myInsertQuery)
    myCommand.Connection = myConnection
    myConnection.Open()
    myCommand.ExecuteNonQuery()
    myCommand.Connection.Close()
End Sub
```

C# example:

```
public void InsertRow(string myConnectionString)
{
    // If the connection string is null, use a default.
    if(myConnectionString == "")
    {
        myConnectionString = "Database=Test;Data Source=localhost;User I
    }
    MySqlConnection myConnection = new MySqlConnection(myConnectionStr
    string myInsertQuery = "INSERT INTO Orders (id, customerId, amount
    MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
    myCommand.Connection = myConnection;
    myConnection.Open();
    myCommand.ExecuteNonQuery();
    myCommand.Connection.Close();
}
```

23.2.3.1.1. Class MySqlCommand Constructor Form 1

Overload methods for MySqlCommand

Initializes a new instance of the MySqlCommand class.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Note. This example shows how to use one of the overloaded versions of the MySqlCommand constructor. For other examples that might be available, see the individual overload topics.

Visual Basic example:

```
Public Sub CreateMySqlCommand()  
    Dim myConnection As New MySqlConnection _  
        ("Persist Security Info=False;database=test;server=myServer"  
        myConnection.Open()  
    Dim myTrans As MySqlTransaction = myConnection.BeginTransaction()  
    Dim mySelectQuery As String = "SELECT * FROM MyTable"  
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection, m  
        myCommand.CommandTimeout = 20  
End Sub
```

C# example:

```
public void CreateMySqlCommand()  
{  
    MySqlConnection myConnection = new MySqlConnection("Persist Secu  
        database=test;server=myServer");  
    myConnection.Open();  
    MySqlTransaction myTrans = myConnection.BeginTransaction();  
    string mySelectQuery = "SELECT * FROM myTable";  
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConne  
        myCommand.CommandTimeout = 20;  
}
```

C++ example:

```
public:  
void CreateMySqlCommand()  
{
```

```

MySQLConnection* myConnection = new MySQLConnection(S"Persist Se
    database=test;server=myServer");
myConnection->Open();
MySQLTransaction* myTrans = myConnection->BeginTransaction();
String* mySelectQuery = S"SELECT * FROM myTable";
MySQLCommand* myCommand = new MySQLCommand(mySelectQuery, myConn
myCommand->CommandTimeout = 20;
};

```

Initializes a new instance of the MySQLCommand class.

The base constructor initializes all fields to their default values. The following table shows initial property values for an instance of MySQLCommand.

Properties	Initial Value
CommandText	empty string ("")
CommandTimeout	0
CommandType	CommandType.Text
Connection	Null

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a MySQLCommand and sets some of its properties.

Visual Basic example:

```

Public Sub CreateMySQLCommand()
    Dim myCommand As New MySQLCommand()
    myCommand.CommandType = CommandType.Text
End Sub

```

C# example:

```

public void CreateMySQLCommand()
{
    MySQLCommand myCommand = new MySQLCommand();
    myCommand.CommandType = CommandType.Text;
}

```

23.2.3.1.2. Class MySqlCommand Constructor Form 2

Initializes a new instance of the MySqlCommand class with the text of the query.

Parameters: The text of the query.

When an instance of MySqlCommand is created, the following read/write properties are set to initial values.

Properties	Initial Value
CommandText	cmdText
CommandTimeout	0
CommandType	CommandType.Text
Connection	Null

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()  
    Dim sql as String = "SELECT * FROM mytable"  
    Dim myCommand As New MySqlCommand(sql)  
    myCommand.CommandType = CommandType.Text  
End Sub
```

C# example:

```
public void CreateMySqlCommand()  
{  
    string sql = "SELECT * FROM mytable";  
    MySqlCommand myCommand = new MySqlCommand(sql);  
    myCommand.CommandType = CommandType.Text;  
}
```

23.2.3.1.3. Class MySqlCommand Constructor Form 3

Initializes a new instance of the MySqlCommand class with the text of the query and a MySqlConnection.

Parameters: The text of the query.

Parameters: A MySqlConnection that represents the connection to an instance of SQL Server.

When an instance of MySqlCommand is created, the following read/write properties are set to initial values.

Properties	Initial Value
CommandText	cmdText
CommandTimeout	0
CommandType	CommandType.Text
Connection	connection

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()  
    Dim conn as new MySqlConnection("server=myServer")  
    Dim sql as String = "SELECT * FROM mytable"  
    Dim myCommand As New MySqlCommand(sql, conn)  
    myCommand.CommandType = CommandType.Text  
End Sub
```

C# example:

```
public void CreateMySqlCommand()  
{  
    MySqlConnection conn = new MySqlConnection("server=myserver")  
    string sql = "SELECT * FROM mytable";  
    MySqlCommand myCommand = new MySqlCommand(sql, conn);  
    myCommand.CommandType = CommandType.Text;  
}
```

23.2.3.1.4. Class MySqlCommand Constructor Form 4

Initializes a new instance of the MySqlCommand class with the text of the query, a MySqlConnection, and the MySqlTransaction.

Parameters: The text of the query.

Parameters: A MySqlConnection that represents the connection to an instance of SQL Server.

Parameters: The MySqlTransaction in which the MySqlCommand executes.

When an instance of MySqlCommand is created, the following read/write properties are set to initial values.

Properties	Initial Value
CommandText	cmdText
CommandTimeout	0
CommandType	CommandType.Text
Connection	connection

You can change the value for any of these properties through a separate call to the property.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySqlCommand()  
    Dim conn as new MySqlConnection("server=myServer")  
    conn.Open();  
    Dim txn as MySqlTransaction = conn.BeginTransaction()  
    Dim sql as String = "SELECT * FROM mytable"  
    Dim myCommand As New MySqlCommand(sql, conn, txn)  
    myCommand.CommandType = CommandType.Text  
End Sub
```

C# example:

```

public void CreateMySqlCommand()
{
    MySqlConnection conn = new MySqlConnection("server=myserver")
    conn.Open();
    MySqlTransaction txn = conn.BeginTransaction();
    string sql = "SELECT * FROM mytable";
    MySqlCommand myCommand = new MySqlCommand(sql, conn, txn);
    myCommand.CommandType = CommandType.Text;
}

```

23.2.3.1.5. ExecuteNonQuery

Executes a SQL statement against the connection and returns the number of rows affected.

Returns: Number of rows affected

You can use ExecuteNonQuery to perform any type of database operation, however any resultsets returned will not be available. Any output parameters used in calling a stored procedure will be populated with data and can be retrieved after execution is complete. For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command. For all other types of statements, the return value is -1.

Examples

The following example creates a MySqlCommand and then executes it using ExecuteNonQuery. The example is passed a string that is a SQL statement (such as UPDATE, INSERT, or DELETE) and a string to use to connect to the data source.

Visual Basic example:

```

Public Sub CreateMySqlCommand(myExecuteQuery As String, myConnec
    Dim myCommand As New MySqlCommand(myExecuteQuery, myConnection
    myCommand.Connection.Open()
    myCommand.ExecuteNonQuery()
    myConnection.Close()
End Sub

```

C# example:

```
public void CreateMySQLCommand(string myExecuteQuery, MySqlConnection
{
    MySqlCommand myCommand = new MySqlCommand(myExecuteQuery, myCo
myCommand.Connection.Open();
myCommand.ExecuteNonQuery();
myConnection.Close();
}
```

23.2.3.1.6. ExecuteReader1

Sends the `CommandText` to the `MySqlConnection`, and builds a `MySqlDataReader` using one of the `CommandBehavior` values.

Parameters: One of the `CommandBehavior` values.

When the `CommandType` property is set to `StoredProcedure`, the `CommandText` property should be set to the name of the stored procedure. The command executes this stored procedure when you call `ExecuteReader`.

The `MySqlDataReader` supports a special mode that enables large binary values to be read efficiently. For more information, see the `SequentialAccess` setting for `CommandBehavior`.

While the `MySqlDataReader` is in use, the associated `MySqlConnection` is busy serving the `MySqlDataReader`. While in this state, no other operations can be performed on the `MySqlConnection` other than closing it. This is the case until the `MySqlDataReader.Close` method of the `MySqlDataReader` is called. If the `MySqlDataReader` is created with `CommandBehavior` set to `CloseConnection`, closing the `MySqlDataReader` closes the connection automatically.

Note. When calling `ExecuteReader` with the `SingleRow` behavior, you should be aware that using a `limit` clause in your SQL will cause all rows (up to the limit given) to be retrieved by the client. The `MySqlDataReader.Read` method will still return `false` after the first row but pulling all rows of data into the client will have a performance impact. If the `limit` clause is not necessary, it should be avoided.

Returns: A `MySqlDataReader` object.

23.2.3.1.7. ExecuteReader

Sends the CommandText to the MySqlConnectionConnection and builds a MySqlDataReader.

Returns: A MySqlDataReader object.

When the CommandType property is set to StoredProcedure, the CommandText property should be set to the name of the stored procedure. The command executes this stored procedure when you call ExecuteReader.

While the MySqlDataReader is in use, the associated MySqlConnection is busy serving the MySqlDataReader. While in this state, no other operations can be performed on the MySqlConnection other than closing it. This is the case until the MySqlDataReader.Close method of the MySqlDataReader is called.

Examples

The following example creates a MySqlCommand, then executes it by passing a string that is a SQL SELECT statement, and a string to use to connect to the data source.

Visual Basic example:

```
Public Sub CreateMySqlDataReader(mySelectQuery As String, myConnecti
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySqlDataReader
    myReader = myCommand.ExecuteReader()
    Try
        While myReader.Read()
            Console.WriteLine(myReader.GetString(0))
        End While
    Finally
        myReader.Close
        myConnection.Close
    End Try
End Sub
```

C# example:

```
public void CreateMySqlDataReader(string mySelectQuery, MySqlConnection
{
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConne
    myConnection.Open();
    MySqlDataReader myReader;
```

```

myReader = myCommand.ExecuteReader();
try
{
    while(myReader.Read())
    {
        Console.WriteLine(myReader.GetString(0));
    }
}
finally
{
    myReader.Close();
    myConnection.Close();
}
}

```

23.2.3.1.8. Prepare

Creates a prepared version of the command on an instance of MySQL Server.

Prepared statements are only supported on MySQL version 4.1 and higher. Calling prepare while connected to earlier versions of MySQL will succeed but will execute the statement in the same way as unprepared.

Examples

The following example demonstrates the use of the Prepare method.

Visual Basic example:

```

public sub PrepareExample()
    Dim cmd as New MySqlCommand("INSERT INTO mytable VALUES (?val)",
    cmd.Parameters.Add( "?val", 10 )
    cmd.Prepare()
    cmd.ExecuteNonQuery()

    cmd.Parameters(0).Value = 20
    cmd.ExecuteNonQuery()
end sub

```

C# example:

```

private void PrepareExample()
{
    MySqlCommand cmd = new MySqlCommand("INSERT INTO mytable VALUES

```

```

cmd.Parameters.Add( "?val", 10 );
cmd.Prepare();
cmd.ExecuteNonQuery();

cmd.Parameters[0].Value = 20;
cmd.ExecuteNonQuery();
}

```

23.2.3.1.9. ExecuteScalar

Executes the query, and returns the first column of the first row in the result set returned by the query. Extra columns or rows are ignored.

Returns: The first column of the first row in the result set, or a null reference if the result set is empty

Use the ExecuteScalar method to retrieve a single value (for example, an aggregate value) from a database. This requires less code than using the ExecuteReader method, and then performing the operations necessary to generate the single value using the data returned by a MySqlDataReader

A typical ExecuteScalar query can be formatted as in the following C# example:

C# example:

```

cmd.CommandText = "select count(*) from region";
Int32 count = (int32) cmd.ExecuteScalar();

```

Examples

The following example creates a MySqlCommand and then executes it using ExecuteScalar. The example is passed a string that is a SQL statement that returns an aggregate result, and a string to use to connect to the data source.

Visual Basic example:

```

Public Sub CreateMySqlCommand(myScalarQuery As String, myConnection
    Dim myCommand As New MySqlCommand(myScalarQuery, myConnection)
    myCommand.Connection.Open()
    myCommand.ExecuteScalar()
    myConnection.Close()

```

End Sub

C# example:

```
public void CreateMySQLCommand(string myScalarQuery, MySqlConnection
{
    MySqlCommand myCommand = new MySqlCommand(myScalarQuery, myConne
myCommand.Connection.Open();
myCommand.ExecuteScalar();
myConnection.Close();
}
```

C++ example:

```
public:
void CreateMySQLCommand(String* myScalarQuery, MySqlConnection*
{
    MySqlCommand* myCommand = new MySqlCommand(myScalarQuery, my
myCommand->Connection->Open();
myCommand->ExecuteScalar();
myConnection->Close();
}
```

23.2.3.1.10. CommandText

Gets or sets the SQL statement to execute at the data source.

Value: The SQL statement or stored procedure to execute. The default is an empty string.

When the CommandType property is set to StoredProcedure, the CommandText property should be set to the name of the stored procedure. The user may be required to use escape character syntax if the stored procedure name contains any special characters. The command executes this stored procedure when you call one of the Execute methods.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySQLCommand()
    Dim myCommand As New MySqlCommand()
```

```
        myCommand.CommandText = "SELECT * FROM Mytable ORDER BY id"
        myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySqlCommand()
{
    MySqlCommand myCommand = new MySqlCommand();
    myCommand.CommandText = "SELECT * FROM mytable ORDER BY id";
    myCommand.CommandType = CommandType.Text;
}
```

23.2.3.1.11. CommandTimeout

Gets or sets the wait time before terminating the attempt to execute a command and generating an error.

Value: The time (in seconds) to wait for the command to execute. The default is 0 seconds.

MySQL currently does not support any method of canceling a pending or executing operation. All commands issues against a MySQL server will execute until completion or exception occurs.

23.2.3.1.12. CommandType

Gets or sets a value indicating how the CommandText property is to be interpreted.

Value: One of the System.Data.CommandType values. The default is Text.

When you set the CommandType property to StoredProcedure, you should set the CommandText property to the name of the stored procedure. The command executes this stored procedure when you call one of the Execute methods.

Examples

The following example creates a MySqlCommand and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySQLCommand()
    Dim myCommand As New MySQLCommand()
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySQLCommand()
{
    MySQLCommand myCommand = new MySQLCommand();
    myCommand.CommandType = CommandType.Text;
}
```

23.2.3.1.13. Connection

Gets or sets the `MySQLConnection` used by this instance of the `MySQLCommand`.

Value: The connection to a data source. The default value is a null reference (Nothing in Visual Basic).

If you set `Connection` while a transaction is in progress and the `Transaction` property is not null, an `InvalidOperationException` is generated. If the `Transaction` property is not null and the transaction has already been committed or rolled back, `Transaction` is set to null.

Examples

The following example creates a `MySQLCommand` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateMySQLCommand()
    Dim mySelectQuery As String = "SELECT * FROM mytable ORDER BY id"
    Dim myConnectionString As String = "Persist Security Info=False;dat
    Dim myCommand As New MySQLCommand(mySelectQuery)
    myCommand.Connection = New MySQLConnection(myConnectionString)
    myCommand.CommandType = CommandType.Text
End Sub
```

C# example:

```
public void CreateMySQLCommand()
{
    string mySelectQuery = "SELECT * FROM mytable ORDER BY id";
```

```

string myConnectionString = "Persist Security Info=False;database=t
MySQLCommand myCommand = new MySQLCommand(mySelectQuery);
myCommand.Connection = new MySqlConnection(myConnectionString);
myCommand.CommandType = CommandType.Text;
}

```

23.2.3.1.14. IsPrepared

Returns true if the statement is prepared.

23.2.3.1.15. Parameters

Get the `MySQLParameterCollection`

Value: The parameters of the SQL statement or stored procedure. The default is an empty collection.

Connector/Net does not support unnamed parameters. Every parameter added to the collection must have an associated name.

Examples

The following example creates a `MySQLCommand` and displays its parameters. To accomplish this, the method is passed a `MySQLConnection`, a query string that is a SQL SELECT statement, and an array of `MySQLParameter` objects.

Visual Basic example:

```

Public Sub CreateMySQLCommand(myConnection As MySqlConnection, _
mySelectQuery As String, myParamArray() As MySQLParameter)
    Dim myCommand As New MySQLCommand(mySelectQuery, myConnection)
    myCommand.CommandText = "SELECT id, name FROM mytable WHERE age="
    myCommand.UpdatedRowSource = UpdateRowSource.Both
    myCommand.Parameters.Add(myParamArray)
    Dim j As Integer
    For j = 0 To myCommand.Parameters.Count - 1
        myCommand.Parameters.Add(myParamArray(j))
    Next j
    Dim myMessage As String = ""
    Dim i As Integer
    For i = 0 To myCommand.Parameters.Count - 1
        myMessage += myCommand.Parameters(i).ToString() & ControlCha
    Next i

```

```
        Console.WriteLine(myMessage)
    End Sub
```

C# example:

```
public void CreateSqlCommand(MySqlConnection myConnection, string
    MySqlParameter[] myParamArray)
{
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery, myConnec
    myCommand.CommandText = "SELECT id, name FROM mytable WHERE age=?
    myCommand.Parameters.Add(myParamArray);
    for (int j=0; j<myParamArray.Length; j++)
    {
        myCommand.Parameters.Add(myParamArray[j]) ;
    }
    string myMessage = "";
    for (int i = 0; i < myCommand.Parameters.Count; i++)
    {
        myMessage += myCommand.Parameters[i].ToString() + "\n";
    }
    MessageBox.Show(myMessage);
}
```

23.2.3.1.16. Transaction

Gets or sets the `MySqlConnection` within which the `MySqlCommand` executes.

Value: The `MySqlConnection`. The default value is a null reference (Nothing in Visual Basic).

You cannot set the `Transaction` property if it is already set to a specific value, and the command is in the process of executing. If you set the transaction property to a `MySqlConnection` object that is not connected to the same `MySqlConnection` as the `MySqlCommand` object, an exception will be thrown the next time you attempt to execute a statement.

23.2.3.1.17. UpdatedRowSource

Gets or sets how command results are applied to the `DataRow` when used by the `System.Data.Common.DbDataAdapter.Update` method of the `System.Data.Common.DbDataAdapter`.

Value: One of the `UpdateRowSource` values.

The default `System.Data.UpdateRowSource` value is `Both` unless the command is automatically generated (as in the case of the `MySqlCommandBuilder`), in which case the default is `None`.

23.2.3.2. MySqlCommandBuilder

Automatically generates single-table commands used to reconcile changes made to a `DataSet` with the associated MySQL database. This class cannot be inherited.

The `MySqlDataAdapter` does not automatically generate the SQL statements required to reconcile changes made to a `System.Data.DataSet` with the associated instance of MySQL. However, you can create a `MySqlCommandBuilder` object to automatically generate SQL statements for single-table updates if you set the `MySqlDataAdapter.SelectCommand` property of the `MySqlDataAdapter`. Then, any additional SQL statements that you do not set are generated by the `MySqlCommandBuilder`.

The `MySqlCommandBuilder` registers itself as a listener for `MySqlDataAdapter.OnRowUpdating` events whenever you set the `DataAdapter` property. You can only associate one `MySqlDataAdapter` or `MySqlCommandBuilder` object with each other at one time.

To generate `INSERT`, `UPDATE`, or `DELETE` statements, the `MySqlCommandBuilder` uses the `SelectCommand` property to retrieve a required set of metadata automatically. If you change the `SelectCommand` after the metadata has been retrieved (for example, after the first update), you should call the `RefreshSchema` method to update the metadata.

The `SelectCommand` must also return at least one primary key or unique column. If none are present, an `InvalidOperationException` exception is generated, and the commands are not generated.

The `MySqlCommandBuilder` also uses the `MySqlCommand.Connection`, `MySqlCommand.CommandTimeout`, and `MySqlCommand.Transaction` properties referenced by the `SelectCommand`. The user should call `RefreshSchema` if any of these properties are modified, or if the `SelectCommand` itself is replaced. Otherwise the

MySqlCommandAdapter.InsertCommandInsertCommand, MySqlCommandAdapter.UpdateCommandUpdateCommand, and MySqlCommandAdapter.DeleteCommandDeleteCommand properties retain their previous values.

If you call Dispose, the MySqlCommandBuilder is disassociated from the MySqlCommandAdapter, and the generated commands are no longer used.

Note. Caution must be used when using MySqlCommandBuilder on MySQL 4.0 systems. With MySQL 4.0, database/schema information is not provided to the connector for a query. This means that a query that pulls columns from two identically named tables in two or more different databases will not cause an exception to be thrown but will not work correctly. Even more dangerous is the situation where your select statement references database X but is executed in database Y and both databases have tables with similar layouts. This situation can cause unwanted changes or deletes. This note does not apply to MySQL versions 4.1 and later.

Examples

The following example uses the MySqlCommand, along MySqlCommandAdapter and MySqlConnection, to select rows from a data source. The example is passed an initialized System.Data.DataSet, a connection string, a query string that is a SQL SELECT statement, and a string that is the name of the database table. The example then creates a MySqlCommandBuilder.

Visual Basic example:

```
Public Shared Function SelectRows(myConnection As String, mySelect
    Dim myConn As New MySqlConnection(myConnection)
    Dim myDataAdapter As New MySqlCommandAdapter()
    myDataAdapter.SelectCommand = New MySqlCommand(mySelectQuery, my
    Dim cb As SqlCommandBuilder = New MySqlCommandBuilder(myDataAdap
    myConn.Open()
    Dim ds As DataSet = New DataSet
    myDataAdapter.Fill(ds, myTableName)
    ' Code to modify data in DataSet here
    ' Without the MySqlCommandBuilder this line would fail.
    myDataAdapter.Update(ds, myTableName)
    myConn.Close()
End Function 'SelectRows
```

C# example:

```
public static DataSet SelectRows(string myConnection, string mySel
{
    MySqlConnection myConn = new MySqlConnection(myConnection);
    MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();
    myDataAdapter.SelectCommand = new MySqlCommand(mySelectQuery, myCo
    MySqlCommandBuilder cb = new MySqlCommandBuilder(myDataAdapter);
    myConn.Open();
    DataSet ds = new DataSet();
    myDataAdapter.Fill(ds, myTableName);
    //code to modify data in DataSet here
    //Without the MySqlCommandBuilder this line would fail
    myDataAdapter.Update(ds, myTableName);
    myConn.Close();
    return ds;
}
```

23.2.3.2.1. Class MySqlCommandBuilder Constructor

Initializes a new instance of the MySqlCommandBuilder class.

23.2.3.2.2. Class MySqlCommandBuilder Constructor Form 1

Initializes a new instance of the MySqlCommandBuilder class and sets the lastOneWins property.

Parameters: False to generate change protection code. True otherwise.

The lastOneWins parameter indicates whether SQL code should be included with the generated DELETE and UPDATE commands that checks the underlying data for changes. If lastOneWins is true then this code is not included and data records could be overwritten in a multi-user or multi-threaded environments. Setting lastOneWins to false will include this check which will cause a concurrency exception to be thrown if the underlying data record has changed without our knowledge.

23.2.3.2.3. Class MySqlCommandBuilder Constructor Form 2

Initializes a new instance of the MySqlCommandBuilder class with the associated MySqlDataAdapter object.

Parameters: The `MySqlDataAdapter` to use.

The `MySqlCommandBuilder` registers itself as a listener for `MySqlDataAdapter.RowUpdating` events that are generated by the `MySqlDataAdapter` specified in this property.

When you create a new instance `MySqlCommandBuilder`, any existing `MySqlCommandBuilder` associated with this `MySqlDataAdapter` is released.

23.2.3.2.4. Class `MySqlCommandBuilder` Constructor Form 3

Initializes a new instance of the `MySqlCommandBuilder` class with the associated `MySqlDataAdapter` object.

Parameters: The `MySqlDataAdapter` to use.

Parameters: False to generate change protection code. True otherwise.

The `MySqlCommandBuilder` registers itself as a listener for `MySqlDataAdapter.RowUpdating` events that are generated by the `MySqlDataAdapter` specified in this property.

When you create a new instance `MySqlCommandBuilder`, any existing `MySqlCommandBuilder` associated with this `MySqlDataAdapter` is released.

The `lastOnWins` parameter indicates whether SQL code should be included with the generated `DELETE` and `UPDATE` commands that checks the underlying data for changes. If `lastOnWins` is true then this code is not included and data records could be overwritten in a multi-user or multi-threaded environments. Setting `lastOnWins` to false will include this check which will cause a concurrency exception to be thrown if the underlying data record has changed without our knowledge.

23.2.3.2.5. `DataAdapter`

Gets or sets a `MySqlDataAdapter` object for which SQL statements are automatically generated.

Value: A `MySqlDataAdapter` object.

The `MySQLCommandBuilder` registers itself as a listener for `MySQLDataAdapter.RowUpdating` events that are generated by the `MySQLDataAdapter` specified in this property.

When you create a new instance `MySQLCommandBuilder`, any existing `MySQLCommandBuilder` associated with this `MySQLDataAdapter` is released.

23.2.3.2.6. QuotePrefix

Gets or sets the beginning character or characters to use when specifying MySQL database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

Value: The beginning character or characters to use. The default value is `.`.

Database objects in MySQL can contain special characters such as spaces that would make normal SQL strings impossible to correctly parse. Use of the `QuotePrefix` and the `QuoteSuffix` properties allows the `MySQLCommandBuilder` to build SQL commands that handle this situation.

23.2.3.2.7. QuoteSuffix

Gets or sets the beginning character or characters to use when specifying MySQL database objects (for example, tables or columns) whose names contain characters such as spaces or reserved tokens.

Value: The beginning character or characters to use. The default value is `.`.

Database objects in MySQL can contain special characters such as spaces that would make normal SQL strings impossible to correctly parse. Use of the `QuotePrefix` and the `QuoteSuffix` properties allows the `MySQLCommandBuilder` to build SQL commands that handle this situation.

23.2.3.2.8. DeriveParameters

23.2.3.2.9. GetDeleteCommand

Gets the automatically generated `MySQLCommand` object required to perform deletions on the database.

Returns: The MySqlCommand object generated to handle delete operations.

An application can use the GetDeleteCommand method for informational or troubleshooting purposes because it returns the MySqlCommand object to be executed.

You can also use GetDeleteCommand as the basis of a modified command. For example, you might call GetDeleteCommand and modify the MySqlCommand.CommandTimeout value, and then explicitly set that on the MySqlDataAdapter.

After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetDeleteCommand will be still be using information from the previous statement, which might not be correct. The SQL statements are first generated either when the application calls System.Data.Common.DataAdapter.Update or GetDeleteCommand.

23.2.3.2.10. GetInsertCommand

Gets the automatically generated MySqlCommand object required to perform insertions on the database.

Returns: The MySqlCommand object generated to handle insert operations.

An application can use the GetInsertCommand method for informational or troubleshooting purposes because it returns the MySqlCommand object to be executed.

You can also use the GetInsertCommand as the basis of a modified command. For example, you might call GetInsertCommand and modify the MySqlCommand.CommandTimeout value, and then explicitly set that on the MySqlDataAdapter.

After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetInsertCommand will be still be using information from the previous statement, which might not be correct. The SQL statements are first generated either when the application calls System.Data.Common.DataAdapter.Update or GetInsertCommand.

23.2.3.2.11. GetUpdateCommand

Gets the automatically generated MySqlCommand object required to perform updates on the database.

Returns: The MySqlCommand object generated to handle update operations.

An application can use the GetUpdateCommand method for informational or troubleshooting purposes because it returns the MySqlCommand object to be executed.

You can also use GetUpdateCommand as the basis of a modified command. For example, you might call GetUpdateCommand and modify the MySqlCommand.CommandTimeout value, and then explicitly set that on the MySqlDataAdapter.

After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetUpdateCommand will be still be using information from the previous statement, which might not be correct. The SQL statements are first generated either when the application calls System.Data.Common.DataAdapter.Update or GetUpdateCommand.

23.2.3.2.12. RefreshSchema

Refreshes the database schema information used to generate INSERT, UPDATE, or DELETE statements.

An application should call RefreshSchema whenever the SELECT statement associated with the MySqlCommandBuilder changes.

An application should call RefreshSchema whenever the MySqlDataAdapter.SelectCommand value of the MySqlDataAdapter changes.

23.2.3.3. MySqlConnection

Represents an open connection to a MySQL Server database. This class cannot be inherited.

A `MySqlConnection` object represents a session to a MySQL Server data source. When you create an instance of `MySqlConnection`, all properties are set to their initial values. For a list of these values, see the `MySqlConnection` constructor.

If the `MySqlConnection` goes out of scope, it is not closed. Therefore, you must explicitly close the connection by calling `MySqlConnection.Close` or `MySqlConnection.Dispose`.

Examples

The following example creates a `MySqlCommand` and a `MySqlConnection`. The `MySqlConnection` is opened and set as the `MySqlCommand.Connection` for the `MySqlCommand`. The example then calls `MySqlCommand.ExecuteNonQuery`, and closes the connection. To accomplish this, the `ExecuteNonQuery` is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
Public Sub InsertRow(myConnectionString As String)
    ' If the connection string is null, use a default.
    If myConnectionString = "" Then
        myConnectionString = "Database=Test;Data Source=localhost;User
    End If
    Dim myConnection As New MySqlConnection(myConnectionString)
    Dim myInsertQuery As String = "INSERT INTO Orders (id, customerI
    Dim myCommand As New MySqlCommand(myInsertQuery)
    myCommand.Connection = myConnection
    myConnection.Open()
    myCommand.ExecuteNonQuery()
    myCommand.Connection.Close()
End Sub
```

C# example:

```
public void InsertRow(string myConnectionString)
{
    // If the connection string is null, use a default.
    if(myConnectionString == "")
    {
        myConnectionString = "Database=Test;Data Source=localhost;User
    }
    MySqlConnection myConnection = new MySqlConnection(myConnections
    string myInsertQuery = "INSERT INTO Orders (id, customerId, amou
    MySqlCommand myCommand = new MySqlCommand(myInsertQuery);
```

```

myCommand.Connection = myConnection;
myConnection.Open();
myCommand.ExecuteNonQuery();
myCommand.Connection.Close();
}

```

23.2.3.3.1. Class MySqlConnection Constructor (Default)

Initializes a new instance of the MySqlConnection class.

When a new instance of MySqlConnection is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the ConnectionString property.

Properties	Initial Value
ConnectionString	empty string ("")
ConnectionTimeout	15
Database	empty string ("")
DataSource	empty string ("")
ServerVersion	empty string ("")

You can change the value for these properties only by using the ConnectionString property.

Examples

Overload methods for MySqlConnection

Initializes a new instance of the MySqlConnection class.

23.2.3.3.2. Class MySqlConnection Constructor Form 1

Initializes a new instance of the MySqlConnection class when given a string containing the connection string.

When a new instance of MySqlConnection is created, the read/write properties are set to the following initial values unless they are specifically set using their associated keywords in the ConnectionString property.

Properties	Initial Value
ConnectionString	empty string ("")
ConnectionTimeout	15
Database	empty string ("")
DataSource	empty string ("")
ServerVersion	empty string ("")

You can change the value for these properties only by using the ConnectionString property.

Examples

Parameters: The connection properties used to open the MySQL database.

23.2.3.3.3. Open

Opens a database connection with the property settings specified by the ConnectionString.

Exception: Cannot open a connection without specifying a data source or server.

Exception: A connection-level error occurred while opening the connection.

The MySqlConnection draws an open connection from the connection pool if one is available. Otherwise, it establishes a new connection to an instance of MySQL.

Examples

The following example creates a MySqlConnection, opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySqlConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
        + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

End Sub

C# example:

```
public void CreateMySQLConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}
```

23.2.3.3.4. Database

Gets the name of the current database or the database to be used after a connection is opened.

Returns: The name of the current database or the name of the database to be used after a connection is opened. The default value is an empty string.

The Database property does not update dynamically. If you change the current database using a SQL statement, then this property may reflect the wrong value. If you change the current database using the ChangeDatabase method, this property is updated to reflect the new database.

Examples

The following example creates a MySqlConnection and displays some of its read-only properties.

Visual Basic example:

```
Public Sub CreateMySQLConnection()
    Dim myConnString As String = _
        "Persist Security Info=False;database=test;server=localhost;user
    Dim myConnection As New MySqlConnection( myConnString )
    myConnection.Open()
    MessageBox.Show( "Server Version: " + myConnection.ServerVersion _
        + ControlChars.NewLine + "Database: " + myConnection.Database )
    myConnection.ChangeDatabase( "test2" )
    MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion _
```

```
        + ControlChars.NewLine + "Database: " + myConnection.Database )
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySQLConnection()
{
    string myConnString =
        "Persist Security Info=False;database=test;server=localhost;user
MySQLConnection myConnection = new MySqlConnection( myConnString )
myConnection.Open();
MessageBox.Show( "Server Version: " + myConnection.ServerVersion
    + "\nDatabase: " + myConnection.Database );
myConnection.ChangeDatabase( "test2" );
MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion
    + "\nDatabase: " + myConnection.Database );
myConnection.Close();
}
```

23.2.3.3.5. State

Gets the current state of the connection.

Returns: A bitwise combination of the System.Data.ConnectionState values. The default is Closed.

The allowed state changes are:

- From Closed to Open, using the Open method of the connection object.
- From Open to Closed, using either the Close method or the Dispose method of the connection object.

Examples

The following example creates a MySqlConnection, opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySQLConnection(myConnString As String)
```

```

    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub

```

C# example:

```

public void CreateMySqlConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}

```

23.2.3.3.6. ServerVersion

Gets a string containing the version of the MySQL server to which the client is connected.

Returns: The version of the instance of MySQL.

Exception: The connection is closed.

Examples

The following example creates a MySqlConnection, opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```

Public Sub CreateMySqlConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub

```

C# example:

```
public void CreateMySQLConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
        "\nState: " + myConnection.State.ToString());
    myConnection.Close();
}
```

23.2.3.3.7. Close

Closes the connection to the database. This is the preferred method of closing any open connection.

The `close` method rolls back any pending transactions. It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled.

An application can call `close` more than one time. No exception is generated.

Examples

The following example creates a `MySqlConnection`, opens it, displays some of its properties, then closes the connection.

Visual Basic example:

```
Public Sub CreateMySQLConnection(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()
    MessageBox.Show("ServerVersion: " + myConnection.ServerVersion _
        + ControlChars.Cr + "State: " + myConnection.State.ToString())
    myConnection.Close()
End Sub
```

C# example:

```
public void CreateMySQLConnection(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
```

```

myConnection.Open();
MessageBox.Show("ServerVersion: " + myConnection.ServerVersion +
"\nState: " + myConnection.State.ToString());
myConnection.Close();
}

```

23.2.3.3.8. CreateCommand

Creates and returns a MySqlCommand object associated with the MySqlConnection.

Returns: A MySqlCommand object.

23.2.3.3.9. BeginTransaction

Begins a database transaction.

Returns: An object representing the new transaction.

Exception: Parallel transactions are not supported.

This command is equivalent to the MySQL BEGIN TRANSACTION command.

You must explicitly commit or roll back the transaction using the MySqlTransaction.Commit or MySqlTransaction.Rollback method.

Note. If you do not specify an isolation level, the default isolation level is used. To specify an isolation level with the BeginTransaction method, use the overload that takes the iso parameter.

Examples

The following example creates a MySqlConnection and a MySqlTransaction. It also demonstrates how to use the BeginTransaction, a MySqlTransaction.Commit, and MySqlTransaction.Rollback methods.

Visual Basic example:

```

Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

```

```

Dim myCommand As MySqlCommand = myConnection.CreateCommand()
Dim myTrans As MySqlTransaction

' Start a local transaction
myTrans = myConnection.BeginTransaction()
' Must assign both transaction object and connection
' to Command object for a pending local transaction
myCommand.Connection = myConnection
myCommand.Transaction = myTrans

Try
    myCommand.CommandText = "Insert into Test (id, desc) VALUES (1
    myCommand.ExecuteNonQuery()
    myCommand.CommandText = "Insert into Test (id, desc) VALUES (1
    myCommand.ExecuteNonQuery()
    myTrans.Commit()
    Console.WriteLine("Both records are written to database.")
Catch e As Exception
    Try
        myTrans.Rollback()
    Catch ex As MySqlException
        If Not myTrans.Connection Is Nothing Then
            Console.WriteLine("An exception of type " + ex.GetType().T
                " was encountered while attempting to ro

        End If
    End Try

    Console.WriteLine("An exception of type " + e.GetType().ToStri
        "was encountered while inserting the data.")
    Console.WriteLine("Neither record was written to database.")
Finally
    myConnection.Close()
End Try
End Sub

```

C# example:

```

public void RunTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction

```

```

myCommand.Connection = myConnection;
myCommand.Transaction = myTrans;
try
{
    myCommand.CommandText = "insert into Test (id, desc) VALUES (1
myCommand.ExecuteNonQuery();
myCommand.CommandText = "insert into Test (id, desc) VALUES (1
myCommand.ExecuteNonQuery();
myTrans.Commit();
    Console.WriteLine("Both records are written to database.");
}
catch(Exception e)
{
    try
    {
        myTrans.Rollback();
    }
    catch (SqlException ex)
    {
        if (myTrans.Connection != null)
        {
            Console.WriteLine("An exception of type " + ex.GetType() +
                " was encountered while attempting to ro
        }
    }

    Console.WriteLine("An exception of type " + e.GetType() +
        " was encountered while inserting the data."
    Console.WriteLine("Neither record was written to database.");
}
finally
{
    myConnection.Close();
}
}

```

23.2.3.3.10. BeginTransaction1

Begins a database transaction with the specified isolation level.

Parameters: The isolation level under which the transaction should run.

Returns: An object representing the new transaction.

Exception: Parallel exceptions are not supported.

This command is equivalent to the MySQL BEGIN TRANSACTION command.

You must explicitly commit or roll back the transaction using the `MySQLTransaction.Commit` or `MySQLTransaction.Rollback` method.

Note. If you do not specify an isolation level, the default isolation level is used. To specify an isolation level with the `BeginTransaction` method, use the overload that takes the `iso` parameter.

Examples

The following example creates a `MySQLConnection` and a `MySQLTransaction`. It also demonstrates how to use the `BeginTransaction`, a `MySQLTransaction.Commit`, and `MySQLTransaction.Rollback` methods.

Visual Basic example:

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySQLTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (1"
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into Test (id, desc) VALUES (1"
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Both records are written to database.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySQLException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " + ex.GetType().T
                    " was encountered while attempting to ro

    End If
End Sub
```

```

        End Try

        Console.WriteLine("An exception of type " + e.GetType().ToString()
            "was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    Finally
        myConnection.Close()
    End Try
End Sub

```

C# example:

```

public void RunTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString);
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "insert into Test (id, desc) VALUES (1
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "insert into Test (id, desc) VALUES (1
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (SqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to ro
            }
        }
    }
}

```

```

        Console.WriteLine("An exception of type " + e.GetType() +
            " was encountered while inserting the data."
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

23.2.3.3.11. ChangeDatabase

Changes the current database for an open MySqlConnection.

Parameters: The name of the database to use.

The value supplied in the database parameter must be a valid database name. The database parameter cannot contain a null value, an empty string, or a string with only blank characters.

When you are using connection pooling against MySQL, and you close the connection, it is returned to the connection pool. The next time the connection is retrieved from the pool, the reset connection request executes before the user performs any operations.

Exception: The database name is not valid.

Exception: The connection is not open.

Exception: Cannot change the database.

Examples

The following example creates a MySqlConnection and displays some of its read-only properties.

Visual Basic example:

```

Public Sub CreateMySqlConnection()
    Dim myConnString As String = _
        "Persist Security Info=False;database=test;server=localhost;user
    Dim myConnection As New MySqlConnection( myConnString )

```

```

myConnection.Open()
MessageBox.Show( "Server Version: " + myConnection.ServerVersion _
    + ControlChars.NewLine + "Database: " + myConnection.Database )
myConnection.ChangeDatabase( "test2" )
MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion _
    + ControlChars.NewLine + "Database: " + myConnection.Database )
myConnection.Close()
End Sub

```

C# example:

```

public void CreateMySQLConnection()
{
    string myConnString =
        "Persist Security Info=False;database=test;server=localhost;user
MySQLConnection myConnection = new MySQLConnection( myConnString )
myConnection.Open();
MessageBox.Show( "Server Version: " + myConnection.ServerVersion
    + "\nDatabase: " + myConnection.Database );
myConnection.ChangeDatabase( "test2" );
MessageBox.Show( "ServerVersion: " + myConnection.ServerVersion
    + "\nDatabase: " + myConnection.Database );
myConnection.Close();
}

```

23.2.3.3.12. StateChange

Occurs when the state of the connection changes.

The StateChange event fires whenever the State changes from closed to opened, or from opened to closed. StateChange fires immediately after the MySQLConnection transitions.

If an event handler throws an exception from within the StateChange event, the exception propagates to the caller of the Open or Close method.

The StateChange event is not raised unless you explicitly call Close or Dispose.

The event handler receives an argument of type System.Data.StateChangeEventArgs containing data related to this event. The following StateChangeEventArgs properties provide information specific to this event.

Property	Description
<code>System.Data.StateChangeEventArgs.CurrentState</code>	Gets the new state of the connection. The connection object will be in the new state already when the event is fired.
<code>System.Data.StateChangeEventArgs.OriginalState</code>	Gets the original state of the connection.

23.2.3.3.13. InfoMessage

Occurs when MySQL returns warnings as a result of executing a command or query.

23.2.3.3.14. ConnectionTimeout

Gets the time to wait while trying to establish a connection before terminating the attempt and generating an error.

Exception: The value set is less than 0.

A value of 0 indicates no limit, and should be avoided in a `MySqlConnection.ConnectionString` because an attempt to connect will wait indefinitely.

Examples

The following example creates a `MySqlConnection` and sets some of its properties in the connection string.

Visual Basic example:

```
Public Sub CreateSqlConnection()
    Dim myConnection As New MySqlConnection()
    myConnection.ConnectionString = "Persist Security Info=False;Usern
    myConnection.Open()
End Sub
```

C# example:

```

public void CreateSqlConnection()
{
    MySqlConnection myConnection = new MySqlConnection();
    myConnection.ConnectionString = "Persist Security Info=False;User=
myConnection.Open();
}

```

23.2.3.3.15. ConnectionString

Gets or sets the string used to connect to a MySQL Server database.

The `ConnectionString` returned may not be exactly like what was originally set but will be identical in terms of keyword/value pairs. Security information will not be included unless the `Persist Security Info` value is set to true.

You can use the `ConnectionString` property to connect to a database. The following example illustrates a typical connection string.

```
"Persist Security Info=False;database=MyDB;server=MySQLServer;user i
```

The `ConnectionString` property can be set only when the connection is closed. Many of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated, except when an error is detected. In this case, none of the properties are updated. `MySqlConnection` properties return only those settings contained in the `ConnectionString`.

To connect to a local machine, specify "localhost" for the server. If you do not specify a server, localhost is assumed.

Resetting the `ConnectionString` on a closed connection resets all connection string values (and related properties) including the password. For example, if you set a connection string that includes "Database= MyDb", and then reset the connection string to "Data Source=myserver;User Id=myUser;Password=myPass", the `MySqlConnection.Database` property is no longer set to MyDb.

The connection string is parsed immediately after being set. If errors in syntax are found when parsing, a runtime exception, such as `ArgumentException`, is generated. Other errors can be found only when an attempt is made to open the connection.

The basic format of a connection string consists of a series of keyword/value pairs separated by semicolons. The equal sign (=) connects each keyword and its value. To include values that contain a semicolon, single-quote character, or double-quote character, the value must be enclosed in double quotes. If the value contains both a semicolon and a double-quote character, the value can be enclosed in single quotes. The single quote is also useful if the value begins with a double-quote character. Conversely, the double quote can be used if the value begins with a single quote. If the value contains both single-quote and double-quote characters, the quote character used to enclose the value must be doubled each time it occurs within the value.

To include preceding or trailing spaces in the string value, the value must be enclosed in either single quotes or double quotes. Any leading or trailing spaces around integer, Boolean, or enumerated values are ignored, even if enclosed in quotes. However, spaces within a string literal keyword or value are preserved. Using .NET Framework version 1.1, single or double quotes may be used within a connection string without using delimiters (for example, Data Source= my'Server or Data Source= my"Server), unless a quote character is the first or last character in the value.

To include an equal sign (=) in a keyword or value, it must be preceded by another equal sign. For example, in the hypothetical connection string

```
"key==word=value"
```

the keyword is "key=word" and the value is "value".

If a specific keyword in a keyword= value pair occurs multiple times in a connection string, the last occurrence listed is used in the value set.

Keywords are not case sensitive.

The following table lists the valid names for keyword values within the `ConnectionString`.

Name	Default	Description
Connect Timeout - or-	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and

Connection Timeout		generating an error.
Host -or- Server -or- Data Source -or- DataSource -or- Address -or- Addr -or- Network Address	localhost	The name or network address of the instance of MySQL to which to connect. Multiple hosts can be specified separated by &. This can be useful where multiple MySQL servers are configured for replication and you are not concerned about the precise server you are connecting to. No attempt is made by the provider to synchronize writes to the database so care should be taken when using this option. In Unix environment with Mono, this can be a fully qualified path to MySQL socket filename. With this configuration, the Unix socket will be used instead of TCP/IP socket. Currently only a single socket name can be given so accessing MySQL in a replicated environment using Unix sockets is not currently supported.
Port	3306	The port MySQL is using to listen for connections. Specify -1 for this value to use a named pipe connection (Windows only). This value is ignored if Unix socket is used.
Protocol	socket	Specifies the type of connection to make to the server. Values can be: socket or tcp for a socket connection pipe for a named pipe connection unix for a Unix socket connection memory to use MySQL shared memory
CharSet -or- Character Set		Specifies the character set that should be used to encode all queries sent to the server. Resultsets are still returned in the character set of the data returned.
Logging	false	When true, various pieces of information is output to any configured TraceListeners.
Allow Batch	true	When true, multiple SQL statements can be sent with one command execution. -Note- Starting with MySQL 4.1.1, batch statements should be separated by the server-defined separator character. Commands sent to earlier versions of MySQL should be separated with ';'
		When true, SSL encryption is used for all data sent between the client and server if the server has a

Encrypt	false	certificate installed. Recognized values are true, false, yes, and no. Note This parameter currently has no effect.
Initial Catalog - or- Database	mysql	The name of the database to use initially
Password - or- pwd		The password for the MySQL account being used.
Persist Security Info	false	When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values including the password. Recognized values are true, false, yes, and no.
User Id -or- Username - or- Uid -or- User name		The MySQL login account being used.
Shared Memory Name	MYSQL	The name of the shared memory object to use for communication if the connection protocol is set to memory.
Allow Zero Datetime	false	True to have MySqlCommandReader.GetValue() return a MySqlDateTime for date or datetime columns that have illegal values. False will cause a DateTime object to be returned for legal values and an exception will be thrown for illegal values.
Convert Zero Datetime	false	True to have MySqlCommandReader.GetValue() and MySqlCommandReader.GetDateTime() return DateTime.MinValue for date or datetime columns that have illegal values.
Old Syntax -or- OldSyntax	false	Allows use of '@' symbol as a parameter marker. See MySqlCommand for more info. This is for compatibility only. All future code should be written to use the new '?' parameter marker.
		When set to the name of a named pipe, the

Pipe Name -or- Pipe	mysql	MySqlConnection will attempt to connect to MySQL on that named pipe. This settings only applies to the Windows platform.
------------------------	-------	--

The following table lists the valid names for connection pooling values within the `ConnectionString`. For more information about connection pooling, see [Connection Pooling for the MySql Data Provider](#).

Name	Default	Description
Connection Lifetime	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by <code>Connection Lifetime</code> . This is useful in clustered configurations to force load balancing between a running server and a server just brought online. A value of zero (0) causes pooled connections to have the maximum connection timeout.
Max Pool Size	100	The maximum number of connections allowed in the pool.
Min Pool Size	0	The minimum number of connections allowed in the pool.
Pooling	true	When true, the <code>MySqlConnection</code> object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are true, false, yes, and no.
Reset Pooled Connections -or- ResetConnections -or- ResetPooledConnections	true	Specifies whether a ping and a reset should be sent to the server before a pooled connection is returned. Not resetting will yeild faster connection opens but also will not clear out session items such as temp tables.
Cache Server Configuration -or-		Specifies whether server variables should be updated when a pooled connection is

CacheServerConfiguration -or- CacheServerConfig	false	returned. Turning this one will yeild faster opens but will also not catch any server changes made by other connections.
--	-------	--

When setting keyword or connection pooling values that require a Boolean value, you can use 'yes' instead of 'true', and 'no' instead of 'false'.

Note The MySql Data Provider uses the native socket protocol to communicate with MySQL. Therefore, it does not support the use of an ODBC data source name (DSN) when connecting to MySQL because it does not add an ODBC layer.

CAUTION In this release, the application should use caution when constructing a connection string based on user input (for example when retrieving user ID and password information from a dialog box, and appending it to the connection string). The application should ensure that a user cannot embed extra connection string parameters in these values (for example, entering a password as "validpassword;database=somedb" in an attempt to attach to a different database).

Examples

The following example creates a MySqlConnection and sets some of its properties

Visual Basic example:

```
Public Sub CreateConnection()
    Dim myConnection As New MySqlConnection()
    myConnection.ConnectionString = "Persist Security Info=False;dat
    myConnection.Open()
End Sub 'CreateConnection
```

C# example:

```
public void CreateConnection()
{
    MySqlConnection myConnection = new MySqlConnection();
    myConnection.ConnectionString = "Persist Security Info=False;dat
    myConnection.Open();
}
```

Examples

The following example creates a `MySqlConnection` in Unix environment with Mono installed. MySQL socket filename used in this example is `"/var/lib/mysql/mysql.sock"`. The actual filename depends on your MySQL configuration.

Visual Basic example:

```
Public Sub CreateConnection()  
    Dim myConnection As New MySqlConnection()  
    myConnection.ConnectionString = "database=myDB;server=/var/lib/m  
    myConnection.Open()  
End Sub 'CreateConnection
```

C# example:

```
public void CreateConnection()  
{  
    MySqlConnection myConnection = new MySqlConnection();  
    myConnection.ConnectionString = "database=myDB;server=/var/lib/m  
    myConnection.Open();  
}
```

23.2.3.4. MySqlConnectionDataAdapter

Represents a set of data commands and a database connection that are used to fill a dataset and update a MySQL database. This class cannot be inherited.

The `MySqlConnectionDataAdapter`, serves as a bridge between a `System.Data.DataSet` and MySQL for retrieving and saving data. The `MySqlConnectionDataAdapter` provides this bridge by mapping `DbDataAdapter.Fill`, which changes the data in the `DataSet` to match the data in the data source, and `DbDataAdapter.Update`, which changes the data in the data source to match the data in the `DataSet`, using the appropriate SQL statements against the data source.

When the `MySqlConnectionDataAdapter` fills a `DataSet`, it will create the necessary tables and columns for the returned data if they do not already exist. However, primary

key information will not be included in the implicitly created schema unless the `System.Data.MissingSchemaAction` property is set to `System.Data.MissingSchemaAction.AddWithKey`. You may also have the `MySQLDataAdapter` create the schema of the `DataSet`, including primary key information, before filling it with data using `System.Data.Common.DbDataAdapter.FillSchema`.

`MySQLDataAdapter` is used in conjunction with `MySQLConnection` and `MySQLCommand` to increase performance when connecting to a MySQL database.

The `MySQLDataAdapter` also includes the `MySQLDataAdapter.SelectCommand`, `MySQLDataAdapter.InsertCommand`, `MySQLDataAdapter.DeleteCommand`, `MySQLDataAdapter.UpdateCommand`, and `DataAdapter.TableMappings` properties to facilitate the loading and updating of data.

When an instance of `MySQLDataAdapter` is created, the read/write properties are set to initial values. For a list of these values, see the `MySQLDataAdapter` constructor.

Note. Please be aware that the `DataColumn` class in .NET 1.0 and 1.1 does not allow columns with type of `UInt16`, `UInt32`, or `UInt64` to be autoincrement columns. If you plan to use autoincrement columns with MySQL, you should consider using signed integer columns.

Examples

The following example creates a `MySQLCommand` and a `MySQLConnection`. The `MySQLConnection` is opened and set as the `MySQLCommand.Connection` for the `MySQLCommand`. The example then calls `MySQLCommand.ExecuteNonQuery`, and closes the connection. To accomplish this, the `ExecuteNonQuery` is passed a connection string and a query string that is a SQL INSERT statement.

Visual Basic example:

```
Public Function SelectRows(dataSet As DataSet, connection As String,
    Dim conn As New MySQLConnection(connection)
    Dim adapter As New MySQLDataAdapter()
    adapter.SelectCommand = new MySQLCommand(query, conn)
    adapter.Fill(dataset)
    Return dataset
End Function
```

C# example:

```
public DataSet SelectRows(DataSet dataset, string connection, string q
{
    MySqlConnection conn = new MySqlConnection(connection);
    MySqlDataAdapter adapter = new MySqlDataAdapter();
    adapter.SelectCommand = new MySqlCommand(query, conn);
    adapter.Fill(dataset);
    return dataset;
}
```

23.2.3.4.1. Class MySqlDataAdapter Constructor

Overload methods for MySqlDataAdapter

Initializes a new instance of the MySqlDataAdapter class.

When an instance of MySqlDataAdapter is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a MySqlDataAdapter and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=1
    "database=test")
    Dim da As MySqlDataAdapter = New MySqlDataAdapter
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey
```

```

da.SelectCommand = New MySqlCommand("SELECT id, name FROM mytabl
da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, na
                                "VALUES (?id, ?name)", c
da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id,
                                "WHERE id=?oldId", conn)
da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE i
da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40

da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40
da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5
da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
End Sub

```

C# example:

```

public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost
    MySqlDataAdapter da = new MySqlDataAdapter();
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.SelectCommand = new MySqlCommand("SELECT id, name FROM mytabl
    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, na
                                "VALUES (?id, ?name)", c
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id,
                                "WHERE id=?oldId", conn)
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE i
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
}

```

23.2.3.4.2. Class **MySqlDataAdapter** Constructor Form 1

Initializes a new instance of the **MySqlDataAdapter** class with the specified **MySqlCommand** as the **SelectCommand** property.

Parameters: **MySqlCommand** that is a SQL **SELECT** statement or stored procedure and is set as the **SelectCommand** property of the **MySqlDataAdapter**.

When an instance of `MySqlConnection` is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
<code>MissingMappingAction</code>	<code>MissingMappingAction.Passthrough</code>
<code>MissingSchemaAction</code>	<code>MissingSchemaAction.Add</code>

You can change the value of any of these properties through a separate call to the property.

When `SelectCommand` (or any of the other command properties) is assigned to a previously created `MySqlCommand`, the `MySqlCommand` is not cloned. The `SelectCommand` maintains a reference to the previously created `MySqlCommand` object.

Examples

The following example creates a `MySqlConnection` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlConnection()  
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=1  
    "database=test")  
    Dim cmd as new MySqlCommand("SELECT id, name FROM mytable", conn)  
    Dim da As MySqlDataAdapter = New MySqlDataAdapter(cmd)  
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey  
  
    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, na  
        "VALUES (?id, ?name)", c  
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id,  
        "WHERE id=?oldId", conn)  
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE i  
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "  
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40  
  
    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "  
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40  
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5  
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "  
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()  
{  
    MySqlConnection conn = new MySqlConnection("Data Source=localhost;  
    MySqlCommand cmd = new MySqlCommand("SELECT id, name FROM mytable");  
    MySqlDataAdapter da = new MySqlDataAdapter(cmd);  
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;  
  
    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, name)  
        VALUES (?id, ?name)", conn);  
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id,  
        WHERE id=?oldId", conn);  
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE id=?id");  
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");  
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");  
  
    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");  
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40, "name");  
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5, "oldId");  
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");  
}
```

23.2.3.4.3. Class `MySqlDataAdapter` Constructor Form 2

Initializes a new instance of the `MySqlDataAdapter` class with a `SelectCommand` and a `MySqlConnection` object.

Parameters: A String that is a SQL SELECT statement or stored procedure to be used by the `SelectCommand` property of the `MySqlDataAdapter`.

Parameters: A `MySqlConnection` that represents the connection.

This implementation of the `MySqlDataAdapter` opens and closes a `MySqlConnection` if it is not already open. This can be useful in an application that must call the `DbDataAdapter.Fill` method for two or more `MySqlDataAdapter` objects. If the `MySqlConnection` is already open, you must explicitly call `MySqlConnection.Close` or `MySqlConnection.Dispose` to close it.

When an instance of `MySqlDataAdapter` is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
------------	---------------

MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a `MySqlDataAdapter` and sets some of its properties.

Visual Basic example:

```
Public Sub CreateSqlDataAdapter()
    Dim conn As MySqlConnection = New MySqlConnection("Data Source=1
    "database=test")
    Dim da As MySqlDataAdapter = New MySqlDataAdapter("SELECT id, na
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

    da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, na
    "VALUES (?id, ?name)", c
    da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id,
    "WHERE id=?oldId", conn)
    da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE i
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
End Sub
```

C# example:

```
public static void CreateSqlDataAdapter()
{
    MySqlConnection conn = new MySqlConnection("Data Source=localhost
    MySqlDataAdapter da = new MySqlDataAdapter("SELECT id, name FROM
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, na
    "VALUES (?id, ?name)", c
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id,
    "WHERE id=?oldId", conn)
```

```

da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE i
da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40

da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40
da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5
da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
}

```

23.2.3.4.4. Class MySqlDataAdapter Constructor Form 3

Initializes a new instance of the MySqlDataAdapter class with a SelectCommand and a connection string.

Parameters: A string that is a SQL SELECT statement or stored procedure to be used by the SelectCommand property of the MySqlDataAdapter.

Parameters: The connection string

When an instance of MySqlDataAdapter is created, the following read/write properties are set to the following initial values.

Properties	Initial Value
MissingMappingAction	MissingMappingAction.Passthrough
MissingSchemaAction	MissingSchemaAction.Add

You can change the value of any of these properties through a separate call to the property.

Examples

The following example creates a MySqlDataAdapter and sets some of its properties.

Visual Basic example:

```

Public Sub CreateSqlDataAdapter()
    Dim da As MySqlDataAdapter = New MySqlDataAdapter("SELECT id, na
    Dim conn As MySqlConnection = da.SelectCommand.Connection
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey

```

```

da.InsertCommand = New MySqlCommand("INSERT INTO mytable (id, na
                                "VALUES (?id, ?name)", c
da.UpdateCommand = New MySqlCommand("UPDATE mytable SET id=?id,
                                "WHERE id=?oldId", conn)
da.DeleteCommand = New MySqlCommand("DELETE FROM mytable WHERE i
da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40

da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40
da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5
da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
End Sub

```

C# example:

```

public static void CreateSqlDataAdapter()
{
    MySqlDataAdapter da = new MySqlDataAdapter("SELECT id, name FROM
    MySqlConnection conn = da.SelectCommand.Connection;
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;

    da.InsertCommand = new MySqlCommand("INSERT INTO mytable (id, na
                                "VALUES (?id, ?name)", c
    da.UpdateCommand = new MySqlCommand("UPDATE mytable SET id=?id,
                                "WHERE id=?oldId", conn)
    da.DeleteCommand = new MySqlCommand("DELETE FROM mytable WHERE i
    da.InsertCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
    da.InsertCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40

    da.UpdateCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
    da.UpdateCommand.Parameters.Add("?name", MySqlDbType.VarChar, 40
    da.UpdateCommand.Parameters.Add("?oldId", MySqlDbType.VarChar, 5
    da.DeleteCommand.Parameters.Add("?id", MySqlDbType.VarChar, 5, "
}

```

23.2.3.4.5. DeleteCommand

Gets or sets a SQL statement or stored procedure used to delete records from the data set.

Value: A MySqlCommand used during System.Data.Common.DataAdapter.Update to delete records in the database that correspond to deleted rows in the DataSet.

During System.Data.Common.DataAdapter.Update, if this property is not set

and primary key information is present in the DataSet, the DeleteCommand can be generated automatically if you set the SelectCommand property and use the MySqlCommandBuilder. Then, any additional commands that you do not set are generated by the MySqlCommandBuilder. This generation logic requires key column information to be present in the DataSet.

When DeleteCommand is assigned to a previously created MySqlCommand, the MySqlCommand is not cloned. The DeleteCommand maintains a reference to the previously created MySqlCommand object.

Examples

The following example creates a MySqlDataAdapter and sets the SelectCommand and DeleteCommand properties. It assumes you have already created a MySqlConnection object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection

    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND nar
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the DeleteCommand.
    cmd = New MySqlCommand("DELETE FROM mytable WHERE id=?id", conn)
    parm = cmd.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id")
    parm.SourceVersion = DataRowVersion.Original
    da.DeleteCommand = cmd
    Return da
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection
{
    MySqlDataAdapter da = new MySqlDataAdapter();
    MySqlCommand cmd;
    MySqlParameter parm;
    // Create the SelectCommand.
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND nar
```

```

cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
da.SelectCommand = cmd;
// Create the DeleteCommand.
cmd = new MySqlCommand("DELETE FROM mytable WHERE id=?id", conn);
parm = cmd.Parameters.Add("?id", MySqlDbType.VarChar, 5, "id");
parm.SourceVersion = DataRowVersion.Original;
da.DeleteCommand = cmd;
return da;
}

```

23.2.3.4.6. InsertCommand

Gets or sets a SQL statement or stored procedure used to insert records into the data set.

Value: A MySqlCommand used during System.Data.Common.DataAdapter.Update to insert records into the database that correspond to new rows in the DataSet.

During System.Data.Common.DataAdapter.Update, if this property is not set and primary key information is present in the DataSet, the InsertCommand can be generated automatically if you set the SelectCommand property and use the MySqlCommandBuilder. Then, any additional commands that you do not set are generated by the MySqlCommandBuilder. This generation logic requires key column information to be present in the DataSet.

When InsertCommand is assigned to a previously created MySqlCommand, the MySqlCommand is not cloned. The InsertCommand maintains a reference to the previously created MySqlCommand object.

Note. If execution of this command returns rows, these rows may be added to the DataSet depending on how you set the MySqlCommand.UpdatedRowSource property of the MySqlCommand object.

Examples

The following example creates a MySqlDataAdapter and sets the SelectCommand and InsertCommand properties. It assumes you have already created a MySqlConnection object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection
```

```
    Dim da As MySqlDataAdapter = New MySqlDataAdapter()  
    Dim cmd As MySqlCommand  
    Dim parm As MySqlParameter  
    ' Create the SelectCommand.  
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND nam  
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)  
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)  
    da.SelectCommand = cmd  
    ' Create the InsertCommand.  
    cmd = New MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,  
    cmd.Parameters.Add( "?id", MySqlDbType.VarChar, 15, "id" )  
    cmd.Parameters.Add( "?name", MySqlDbType.VarChar, 15, "name" )  
    da.InsertCommand = cmd
```

```
    Return da  
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection  
{  
    MySqlDataAdapter da = new MySqlDataAdapter();  
    MySqlCommand cmd;  
    MySqlParameter parm;  
    // Create the SelectCommand.  
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND nam  
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);  
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);  
    da.SelectCommand = cmd;  
    // Create the InsertCommand.  
    cmd = new MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,  
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" );  
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" );  
  
    da.InsertCommand = cmd;  
    return da;  
}
```

23.2.3.4.7. UpdateCommand

Gets or sets a SQL statement or stored procedure used to updated records in the data source.

Value: A MySqlCommand used during

System.Data.Common.DataAdapter.Update to update records in the database

with data from the DataSet.

During `System.Data.Common.DataAdapter.Update`, if this property is not set and primary key information is present in the DataSet, the `UpdateCommand` can be generated automatically if you set the `SelectCommand` property and use the `MySqlCommandBuilder`. Then, any additional commands that you do not set are generated by the `MySqlCommandBuilder`. This generation logic requires key column information to be present in the DataSet.

When `UpdateCommand` is assigned to a previously created `MySqlCommand`, the `MySqlCommand` is not cloned. The `UpdateCommand` maintains a reference to the previously created `MySqlCommand` object.

Note. If execution of this command returns rows, these rows may be merged with the DataSet depending on how you set the `MySqlCommand.UpdatedRowSource` property of the `MySqlCommand` object.

Examples

The following example creates a `MySqlDataAdapter` and sets the `SelectCommand` and `UpdateCommand` properties. It assumes you have already created a `MySqlConnection` object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection)

    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND nam
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the UpdateCommand.
    cmd = New MySqlCommand("UPDATE mytable SET id=?id, name=?name WHER
    cmd.Parameters.Add( "?id", MySqlDbType.VarChar, 15, "id" )
    cmd.Parameters.Add( "?name", MySqlDbType.VarChar, 15, "name" )

    parm = cmd.Parameters.Add("?oldId", MySqlDbType.VarChar, 15, "id")
    parm.SourceVersion = DataRowVersion.Original

    da.UpdateCommand = cmd
```

```
Return da  
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection  
{  
    MySqlDataAdapter da = new MySqlDataAdapter();  
    MySqlCommand cmd;  
    MySqlParameter parm;  
    // Create the SelectCommand.  
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND nam  
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);  
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);  
    da.SelectCommand = cmd;  
    // Create the UpdateCommand.  
    cmd = new MySqlCommand("UPDATE mytable SET id=?id, name=?name WHER  
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" );  
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" );  
  
    parm = cmd.Parameters.Add( "?oldId", MySqlDbType.VarChar, 15, "id"  
    parm.SourceVersion = DataRowVersion.Original;  
  
    da.UpdateCommand = cmd;  
    return da;  
}
```

23.2.3.4.8. SelectCommand

Gets or sets a SQL statement or stored procedure used to select records in the data source.

Value: A MySqlCommand used during System.Data.Common.DbDataAdapter.Fill to select records from the database for placement in the DataSet.

When SelectCommand is assigned to a previously created MySqlCommand, the MySqlCommand is not cloned. The SelectCommand maintains a reference to the previously created MySqlCommand object.

If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.

Examples

The following example creates a `MySqlDataAdapter` and sets the `SelectCommand` and `InsertCommand` properties. It assumes you have already created a `MySqlConnection` object.

Visual Basic example:

```
Public Shared Function CreateCustomerAdapter(conn As MySqlConnection

    Dim da As MySqlDataAdapter = New MySqlDataAdapter()
    Dim cmd As MySqlCommand
    Dim parm As MySqlParameter
    ' Create the SelectCommand.
    cmd = New MySqlCommand("SELECT * FROM mytable WHERE id=?id AND nar
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15)
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15)
    da.SelectCommand = cmd
    ' Create the InsertCommand.
    cmd = New MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,
    cmd.Parameters.Add( "?id", MySqlDbType.VarChar, 15, "id" )
    cmd.Parameters.Add( "?name", MySqlDbType.VarChar, 15, "name" )
    da.InsertCommand = cmd

    Return da
End Function
```

C# example:

```
public static MySqlDataAdapter CreateCustomerAdapter(MySqlConnection
{
    MySqlDataAdapter da = new MySqlDataAdapter();
    MySqlCommand cmd;
    MySqlParameter parm;
    // Create the SelectCommand.
    cmd = new MySqlCommand("SELECT * FROM mytable WHERE id=?id AND nar
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15);
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15);
    da.SelectCommand = cmd;
    // Create the InsertCommand.
    cmd = new MySqlCommand("INSERT INTO mytable (id,name) VALUES (?id,
    cmd.Parameters.Add("?id", MySqlDbType.VarChar, 15, "id" );
    cmd.Parameters.Add("?name", MySqlDbType.VarChar, 15, "name" );

    da.InsertCommand = cmd;
    return da;
}
```

23.2.3.5. MySqlDataReader

To create a `MySQLDataReader`, you must call the `MySQLCommand.ExecuteReader` method of the `MySQLCommand` object, rather than directly using a constructor.

While the `MySQLDataReader` is in use, the associated `MySQLConnection` is busy serving the `MySQLDataReader`, and no other operations can be performed on the `MySQLConnection` other than closing it. This is the case until the `MySQLDataReader.Close` method of the `MySQLDataReader` is called.

`MySQLDataReader.IsClosed` and `MySQLDataReader.RecordsAffected` are the only properties that you can call after the `MySQLDataReader` is closed. Though the `RecordsAffected` property may be accessed at any time while the `MySQLDataReader` exists, always call `Close` before returning the value of `RecordsAffected` to ensure an accurate return value.

For optimal performance, `MySQLDataReader` avoids creating unnecessary objects or making unnecessary copies of data. As a result, multiple calls to methods such as `MySQLDataReader.GetValue` return a reference to the same object. Use caution if you are modifying the underlying value of the objects returned by methods such as `GetValue`.

Examples

The following example creates a `MySQLConnection`, a `MySQLCommand`, and a `MySQLDataReader`. The example reads through the data, writing it out to the console. Finally, the example closes the `MySQLDataReader`, then the `MySQLConnection`.

Visual Basic example:

```
Public Sub ReadMyData(myConnString As String)
    Dim mySelectQuery As String = "SELECT OrderID, CustomerID FROM C
    Dim myConnection As New MySqlConnection(myConnString)
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)
    myConnection.Open()
    Dim myReader As MySQLDataReader
    myReader = myCommand.ExecuteReader()
    ' Always call Read before accessing data.
    While myReader.Read()
        Console.WriteLine((myReader.GetInt32(0) & ", " & myReader.Ge
    End While
    ' always call Close when done reading.
    myReader.Close()
    ' Close the connection when done with it.
```

```
        myConnection.Close()
End Sub 'ReadMyData
```

C# example:

```
public void ReadMyData(string myConnString) {
    string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
    MySqlConnection myConnection = new MySqlConnection(myConnString)
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery,myConnec
    myConnection.Open();
    MySqlDataReader myReader;
    myReader = myCommand.ExecuteReader();
    // Always call Read before accessing data.
    while (myReader.Read()) {
        Console.WriteLine(myReader.GetInt32(0) + ", " + myReader.GetS
    }
    // always call Close when done reading.
    myReader.Close();
    // Close the connection when done with it.
    myConnection.Close();
}
```

23.2.3.5.1. GetBytes

getBytes returns the number of available bytes in the field. In most cases this is the exact length of the field. However, the number returned may be less than the true length of the field if GetBytes has already been used to obtain bytes from the field. This may be the case, for example, if the MySqlDataReader is reading a large data structure into a buffer. For more information, see the SequentialAccess setting for MySqlCommand.CommandBehavior.

If you pass a buffer that is a null reference (Nothing in Visual Basic), GetBytes returns the length of the field in bytes.

No conversions are performed; therefore the data retrieved must already be a byte array.

23.2.3.5.2. GetTimeSpan

Gets the value of the specified column as a TimeSpan object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.3. GetDateTime

Gets the value of the specified column as a `DateTime` object.

Note. MySQL allows date columns to contain the value '0000-00-00' and datetime columns to contain the value '0000-00-00 00:00:00'. The `DateTime` structure cannot contain or represent these values. To read a datetime value from a column that might contain zero values, use `GetMySQLDateTime`. The behavior of reading a zero datetime column using this method is defined by the `ZeroDateTimeBehavior` connection string option. For more information on this option, please refer to `MySQLConnection.ConnectionString`.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.4. GetMySQLDateTime

Gets the value of the specified column as a `MySQL.Data.Types.MySQLDateTime` object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.5. GetString

Gets the value of the specified column as a `String` object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.6. GetDecimal

Gets the value of the specified column as a `Decimal` object.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.7. GetDouble

Gets the value of the specified column as a double-precision floating point number.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.8. GetFloat

Gets the value of the specified column as a single-precision floating point number.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.9. GetGuid

Gets the value of the specified column as a globally-unique identifier (GUID).

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.10. GetInt16

Gets the value of the specified column as a 16-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.11. GetInt32

Gets the value of the specified column as a 32-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.12. GetInt64

Gets the value of the specified column as a 64-bit signed integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.13. GetUInt16

Gets the value of the specified column as a 16-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.14. GetUInt32

Gets the value of the specified column as a 32-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.5.15. GetUInt64

Gets the value of the specified column as a 64-bit unsigned integer.

Parameters: The zero-based column ordinal.

Returns: The value of the specified column.

23.2.3.6. MySqlConnectionException

This class is created whenever the MySQL Data Provider encounters an error generated from the server.

Any open connections are not automatically closed when an exception is thrown. If the client application determines that the exception is fatal, it should close any open `MySqlDataReader` objects or `MySqlConnection` objects.

Examples

The following example generates a `MySqlException` due to a missing server, and then displays the exception.

Visual Basic example:

```
Public Sub ShowException()  
    Dim mySelectQuery As String = "SELECT column1 FROM table1"  
    Dim myConnection As New MySqlConnection ("Data Source=localhost  
    Dim myCommand As New MySqlCommand(mySelectQuery, myConnection)  
    Try  
        myCommand.Connection.Open()  
    Catch e As MySqlException  
        MessageBox.Show( e.Message )  
    End Try  
End Sub
```

C# example:

```
public void ShowException()  
{  
    string mySelectQuery = "SELECT column1 FROM table1";  
    MySqlConnection myConnection =  
        new MySqlConnection("Data Source=localhost;Database=Sample;");  
    MySqlCommand myCommand = new MySqlCommand(mySelectQuery,myConnect  
    try  
    {  
        myCommand.Connection.Open();  
    }  
    catch (MySqlException e)  
    {  
        MessageBox.Show( e.Message );  
    }  
}
```

23.2.3.7. MySqlParameter

Parameter names are not case sensitive.

Examples

The following example creates multiple instances of `MySQLParameter` through the `MySQLParameterCollection` collection within the `MySQLDataAdapter`. These parameters are used to select data from the data source and place the data in the `DataSet`. This example assumes that a `DataSet` and a `MySQLDataAdapter` have already been created with the appropriate schema, commands, and connection.

Visual Basic example:

```
Public Sub AddSqlParameter()  
    ' ...  
    ' create myDataSet and myDataAdapter  
    ' ...  
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar)  
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Int32)  
  
    myDataAdapter.Fill(myDataSet)  
End Sub 'AddSqlParameter
```

C# example:

```
public void AddSqlParameter()  
{  
    // ...  
    // create myDataSet and myDataAdapter  
    // ...  
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar)  
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Int32)  
    myDataAdapter.Fill(myDataSet);  
}
```

23.2.3.8. MySQLParameterCollection

The number of the parameters in the collection must be equal to the number of parameter placeholders within the command text, or an exception will be generated.

Examples

The following example creates multiple instances of `MySQLParameter` through the `MySQLParameterCollection` collection within the `MySQLDataAdapter`. These

parameters are used to select data within the data source and place the data in the DataSet. This code assumes that a DataSet and a MySqlDataAdapter have already been created with the appropriate schema, commands, and connection.

Visual Basic example:

```
Public Sub AddParameters()  
    ' ...  
    ' create myDataSet and myDataAdapter  
    ' ...  
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar)  
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Int32)  
  
    myDataAdapter.Fill(myDataSet)  
End Sub 'AddSqlParameters
```

C# example:

```
public void AddSqlParameters()  
{  
    // ...  
    // create myDataSet and myDataAdapter  
    // ...  
    myDataAdapter.SelectCommand.Parameters.Add("@CategoryName", MySqlDbType.VarChar)  
    myDataAdapter.SelectCommand.Parameters.Add("@SerialNum", MySqlDbType.Int32)  
    myDataAdapter.Fill(myDataSet);  
}
```

23.2.3.9. MySqlConnectionTransaction

Represents a SQL transaction to be made in a MySQL database. This class cannot be inherited.

The application creates a MySqlConnectionTransaction object by calling MySqlConnection.BeginTransaction on the MySqlConnection object. All subsequent operations associated with the transaction (for example, committing or aborting the transaction), are performed on the MySqlConnectionTransaction object.

Examples

The following example creates a MySqlConnection and a MySqlConnectionTransaction. It also demonstrates how to use the MySqlConnection.BeginTransaction,

MySQLTransaction.Commit, and MySQLTransaction.Rollback methods.

Visual Basic example:

```
Public Sub RunTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySQLTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()
    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into Region (RegionID, RegionD
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into Region (RegionID, RegionD
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Both records are written to database.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySQLException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " & ex.GetType().T
                    " was encountered while attempting to ro

            End If
        End Try

        Console.WriteLine("An exception of type " & e.GetType().ToStri
            "was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    Finally
        myConnection.Close()
    End Try
End Sub 'RunTransaction
```

C# example:

```
public void RunTransaction(string myConnString)
{
```

```

 MySqlConnection myConnection = new MySqlConnection(myConnString)
 myConnection.Open();
 MySqlCommand myCommand = myConnection.CreateCommand();
 MySqlTransaction myTrans;
 // Start a local transaction
 myTrans = myConnection.BeginTransaction();
 // Must assign both transaction object and connection
 // to Command object for a pending local transaction
 myCommand.Connection = myConnection;
 myCommand.Transaction = myTrans;
 try
 {
     myCommand.CommandText = "Insert into Region (RegionID, RegionD
     myCommand.ExecuteNonQuery();
     myCommand.CommandText = "Insert into Region (RegionID, RegionD
     myCommand.ExecuteNonQuery();
     myTrans.Commit();
     Console.WriteLine("Both records are written to database.");
 }
 catch(Exception e)
 {
     try
     {
         myTrans.Rollback();
     }
     catch (MySqlException ex)
     {
         if (myTrans.Connection != null)
         {
             Console.WriteLine("An exception of type " + ex.GetType() +
                 " was encountered while attempting to ro
         }
     }
 }

 Console.WriteLine("An exception of type " + e.GetType() +
     " was encountered while inserting the data."
 Console.WriteLine("Neither record was written to database.");
 }
 finally
 {
     myConnection.Close();
 }
 }

```

23.2.3.9.1. Rollback

Rolls back a transaction from a pending state.

The Rollback method is equivalent to the MySQL statement ROLLBACK. The transaction can only be rolled back from a pending state (after BeginTransaction has been called, but before Commit is called).

Examples

The following example creates MySqlConnection and a MySqlTransaction. It also demonstrates how to use the MySqlConnection.BeginTransaction, Commit, and Rollback methods.

Visual Basic example:

```
Public Sub RunSqlTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()

    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Success.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySqlException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " & ex.GetType().T
                " was encountered while attempting to ro

            End If
        End Try

        Console.WriteLine("An exception of type " & e.GetType().ToStri
        "was encountered while inserting the data.")
        Console.WriteLine("Neither record was written to database.")
    End Try
End Sub
```

```
        Finally
            myConnection.Close()
        End Try
    End Sub
```

C# example:

```
public void RunSqlTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString)
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (MySqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                " was encountered while attempting to ro
            }
        }

        Console.WriteLine("An exception of type " + e.GetType() +
        " was encountered while inserting the data."
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
```

```

        myConnection.Close();
    }
}

```

23.2.3.9.2. Commit

Commits the database transaction.

The Commit method is equivalent to the MySQL SQL statement COMMIT.

Examples

The following example creates MySqlConnection and a MySqlTransaction. It also demonstrates how to use the MySqlConnection.BeginTransaction, Commit, and Rollback methods.

Visual Basic example:

```

Public Sub RunSqlTransaction(myConnString As String)
    Dim myConnection As New MySqlConnection(myConnString)
    myConnection.Open()

    Dim myCommand As MySqlCommand = myConnection.CreateCommand()
    Dim myTrans As MySqlTransaction

    ' Start a local transaction
    myTrans = myConnection.BeginTransaction()

    ' Must assign both transaction object and connection
    ' to Command object for a pending local transaction
    myCommand.Connection = myConnection
    myCommand.Transaction = myTrans

    Try
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES
        myCommand.ExecuteNonQuery()
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES
        myCommand.ExecuteNonQuery()
        myTrans.Commit()
        Console.WriteLine("Success.")
    Catch e As Exception
        Try
            myTrans.Rollback()
        Catch ex As MySqlException
            If Not myTrans.Connection Is Nothing Then
                Console.WriteLine("An exception of type " & ex.GetType().T

```

```

        " was encountered while attempting to ro
    End If
End Try

    Console.WriteLine("An exception of type " & e.GetType().ToStri
        "was encountered while inserting the data.")
    Console.WriteLine("Neither record was written to database.")
Finally
    myConnection.Close()
End Try
End Sub

```

C# example:

```

public void RunSqlTransaction(string myConnString)
{
    MySqlConnection myConnection = new MySqlConnection(myConnString)
    myConnection.Open();
    MySqlCommand myCommand = myConnection.CreateCommand();
    MySqlTransaction myTrans;
    // Start a local transaction
    myTrans = myConnection.BeginTransaction();
    // Must assign both transaction object and connection
    // to Command object for a pending local transaction
    myCommand.Connection = myConnection;
    myCommand.Transaction = myTrans;
    try
    {
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES
        myCommand.ExecuteNonQuery();
        myCommand.CommandText = "Insert into mytable (id, desc) VALUES
        myCommand.ExecuteNonQuery();
        myTrans.Commit();
        Console.WriteLine("Both records are written to database.");
    }
    catch(Exception e)
    {
        try
        {
            myTrans.Rollback();
        }
        catch (MySqlException ex)
        {
            if (myTrans.Connection != null)
            {
                Console.WriteLine("An exception of type " + ex.GetType() +
                    " was encountered while attempting to ro
            }
        }
    }
}

```

```

        Console.WriteLine("An exception of type " + e.GetType() +
                          " was encountered while inserting the data.");
        Console.WriteLine("Neither record was written to database.");
    }
    finally
    {
        myConnection.Close();
    }
}

```

23.2.4. Connector/NET Reference

This section of the manual contains a complete reference to the Connector/NET ADO.NET component, automatically generated from the embedded documentation.

23.2.4.1. MySql.Data.MySqlClient

[Namespace hierarchy](#)

Classes

Class	Description
MySqlCommand	
MySqlCommandBuilder	
MySqlConnection	
MySqlDataAdapter	
MySqlDataReader	Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.
MySqlError	Collection of error codes that can be returned by the server
MySqlException	The exception that is thrown when MySQL returns an error. This class cannot be inherited.
MySqlHelper	Helper class that makes it easier to work with the provider.
	Provides data for the InfoMessage event. This

MySqlInfoMessageEventArgs	class cannot be inherited.
MySqlParameter	Represents a parameter to a MySqlCommand , and optionally, its mapping to DataSetcolumns. This class cannot be inherited.
MySqlParameterCollection	Represents a collection of parameters relevant to a MySqlCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.
MySqlRowUpdatedEventArgs	Provides data for the RowUpdated event. This class cannot be inherited.
MySqlRowUpdatingEventArgs	Provides data for the RowUpdating event. This class cannot be inherited.
MySqlTransaction	

Delegates

Delegate	Description
MySqlInfoMessageEventHandler	Represents the method that will handle the InfoMessage event of a MySqlConnection .
MySqlRowUpdatedEventHandler	Represents the method that will handle the RowUpdatedevent of a MySqlDataAdapter .
MySqlRowUpdatingEventHandler	Represents the method that will handle the RowUpdatingevent of a MySqlDataAdapter .

Enumerations

Enumeration	Description
MySqlDbType	Specifies MySQL specific data type of a field, property, for use in a MySqlParameter .
MySqlErrorCode	

23.2.4.1.1. MySql.Data.MySqlClientHierarchy

See Also

[MySQL.Data.MySqlCommand Namespace](#)

23.2.4.1.2. MySqlCommand Class

For a list of all members of this type, see [MySqlCommand Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommand_  
    Inherits Component_  
    Implements IDbCommand, ICloneable
```

Syntax: C#

```
public sealed class MySqlCommand : Component, IDbCommand, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlCommand](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySqlCommand Members](#) , [MySQL.Data.MySqlCommand Namespace](#)

23.2.4.1.2.1. MySqlCommand Members

[MySqlCommand overview](#)

Public Instance Constructors

MySqlCommand	Overloaded. Initializes a new instance of the MySqlCommand class.
------------------------------	---

Public Instance Properties

CommandText	
CommandTimeout	
CommandType	
Connection	
Container(inherited from Component)	Gets the IContainerthat contains the Component.
IsPrepared	
Parameters	
Site(inherited from Component)	Gets or sets the ISiteof the Component.
Transaction	
UpdatedRowSource	

Public Instance Methods

Cancel	Attempts to cancel the execution of a MySqlCommand. This operation is not supported.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
CreateParameter	Creates a new instance of a MySqlParameter object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
ExecuteNonQuery	
ExecuteReader	Overloaded.
ExecuteScalar	
GetHashCode(inherited from	Serves as a hash function for a particular type. GetHashCodeis suitable for use in

Object)	hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Prepare	
ToString(inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
------------------------------------	---

See Also

[MySQLCommand Class](#) , [MySQL.Data.MySqlConnection Namespace](#)

23.2.4.1.2.1.1. MySQLCommand Constructor

Initializes a new instance of the [MySQLCommand](#) class.

Overload List

Initializes a new instance of the [MySQLCommand](#) class.

- [public MySQLCommand\(\);](#)
- [public MySQLCommand\(string\);](#)
- [public MySQLCommand\(string, MySqlConnection\);](#)
- [public MySQLCommand\(string, MySqlConnection, MySqlTransaction\);](#)

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1. MySqlCommand Constructor ()

Initializes a new instance of the [MySqlCommand](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommand();
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
 [MySqlCommand Constructor Overload List](#)

23.2.4.1.2.1.1.2. MySqlCommand Constructor (String)

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String _  
)
```

Syntax: C#

```
public MySqlCommand(  
    stringcmdText  
);
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
 [MySqlCommand Constructor Overload List](#)

23.2.4.1.2.1.1.3. MySqlCommand Constructor (String, MySqlConnection)

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection _  
)
```

Syntax: C#

```
public MySqlCommand(  
    string cmdText,  
    MySqlConnection connection  
) ;
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlCommand Constructor Overload List](#)

23.2.4.1.2.1.1.3.1. MySqlConnection Class

For a list of all members of this type, see [MySqlConnection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_  
    Inherits Component_  
    Implements IDbConnection, ICloneable
```

Syntax: C#

```
public sealed class MySqlConnection : Component, IDbConnection, IClo
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlConnection Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1. MySqlConnection Members

[MySqlConnection overview](#)

Public Instance Constructors

MySqlConnection	Overloaded. Initializes a new instance of the MySqlConnection class.
---------------------------------	--

Public Instance Properties

ConnectionString	
ConnectionTimeout	
Container(inherited from Component)	Gets the IContainerthat contains the Component.
Database	
DataSource	Gets the name of the MySQL server to which to connect.
ServerThread	Returns the id of the server thread this connection is executing on
ServerVersion	
Site(inherited from Component)	Gets or sets the ISiteof the Component.
State	
UseCompression	Indicates if this connection should use compression when communicating with the server.

Public Instance Methods

BeginTransaction	Overloaded.
----------------------------------	-------------

ChangeDatabase	
Close	
CreateCommand	
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Open	
Ping	Ping
ToString(inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
InfoMessage	
StateChange	

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.1. MySqlConnection Constructor

Initializes a new instance of the [MySqlConnection](#) class.

Overload List

Initializes a new instance of the [MySqlConnection](#) class.

- [public MySqlConnection\(\);](#)
- [public MySqlConnection\(string\);](#)

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.1. MySqlConnection Constructor ()

Initializes a new instance of the [MySqlConnection](#) class.

Syntax: Visual Basic

Overloads Public Sub New()

Syntax: C#

```
public MySqlConnection();
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
 [MySqlConnection Constructor Overload List](#)

23.2.4.1.2.1.1.3.1.1.2. MySqlConnection Constructor (String)

Syntax: Visual Basic

Overloads Public Sub New(_
 ByVal connectionString As String _

)

Syntax: C#

```
public MySqlConnection(  
stringconnectionString  
)  
;
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlConnection Constructor Overload List](#)

23.2.4.1.2.1.1.3.1.1.2. **ConnectionString** Property

Syntax: Visual Basic

```
NotOverridable Public Property ConnectionString As String _  
— Implements IDbConnection.ConnectionString
```

Syntax: C#

```
public string ConnectionString {get; set;}
```

Implements

IDbConnection.ConnectionString

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.3. **ConnectionTimeout** Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property ConnectionTimeout As Integer  
— Implements IDbConnection.ConnectionTimeout
```

Syntax: C#

```
public int ConnectionTimeout {get;}
```

Implements

IDbConnection.ConnectionTimeout

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.4. Database Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Database As String _  
    Implements IDbConnection.Database
```

Syntax: C#

```
public string Database {get;}
```

Implements

IDbConnection.Database

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.5. DataSource Property

Gets the name of the MySQL server to which to connect.

Syntax: Visual Basic

```
Public ReadOnly Property DataSource As String
```

Syntax: C#

```
public string DataSource {get;}
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.6. ServerThread Property

Returns the id of the server thread this connection is executing on

Syntax: Visual Basic

```
Public ReadOnly Property ServerThread As Integer
```

Syntax: C#

```
public int ServerThread {get;}
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.7. ServerVersion Property

Syntax: Visual Basic

```
Public ReadOnly Property ServerVersion As String
```

Syntax: C#

```
public string ServerVersion {get;}
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.8. State Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property State As ConnectionState _
```

```
— Implements IDbConnection.State
```

Syntax: C#

```
public System.Data.ConnectionState State {get;}
```

Implements

IDbConnection.State

See Also

[MySQLConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.9. UseCompression Property

Indicates if this connection should use compression when communicating with the server.

Syntax: Visual Basic

```
Public ReadOnly Property UseCompression As Boolean
```

Syntax: C#

```
public bool UseCompression {get;}
```

See Also

[MySQLConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.10. BeginTransaction Method

Overload List

- [public MySqlConnection BeginTransaction\(\);](#)
- [public MySqlConnection BeginTransaction\(IsolationLevel\);](#)

See Also

[MySQLConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.10.1. MySqlConnection.BeginTransaction Method ()

Syntax: Visual Basic

Overloads Public Function BeginTransaction() As MySqlTransaction

Syntax: C#

```
public MySqlTransaction BeginTransaction();
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlConnection.BeginTransaction Overload List](#)

23.2.4.1.2.1.1.3.1.1.10.1.1.1. MySqlConnection Class

For a list of all members of this type, see [MySqlConnection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_  
    Implements IDbTransaction, IDisposable
```

Syntax: C#

```
public sealed class MySqlConnection : IDbTransaction, IDisposable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlConnection Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.10.1.1.1. MySqlConnection Members

[MySQLTransaction overview](#)

Public Instance Properties

Connection	Gets the MySQLConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
IsolationLevel	Specifies the IsolationLevel for this transaction.

Public Instance Methods

Commit	
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
Rollback	
ToString(inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLTransaction Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.10.1.1.1.1. Connection Property

Gets the [MySQLConnection](#) object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Syntax: Visual Basic

```
Public ReadOnly Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get;}
```

Property Value

The [MySqlConnection](#) object associated with this transaction.

Remarks

A single application may have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by [BeginTransaction](#) .

See Also

[MySqlTransaction Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.10.1.1.1.2. IsolationLevel Property

Specifies the IsolationLevel for this transaction.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsolationLevel As IsolationL  
- Implements IDbTransaction.IsolationLevel
```

Syntax: C#

```
public System.Data.IsolationLevel IsolationLevel {get;}
```

Property Value

The IsolationLevel for this transaction. The default is ReadCommitted.

Implements

IDbTransaction.IsolationLevel

Remarks

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.

See Also

[MySQLTransaction Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.10.1.1.1.3. MySQLTransaction.Commit Method

Syntax: Visual Basic

```
NotOverridable Public Sub Commit() _  
    Implements IDbTransaction.Commit
```

Syntax: C#

```
public void Commit();
```

Implements

IDbTransaction.Commit

See Also

[MySQLTransaction Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.10.1.1.1.4. MySQLTransaction.Rollback Method

Syntax: Visual Basic

```
NotOverridable Public Sub Rollback() _  
    Implements IDbTransaction.Rollback
```

Syntax: C#

```
public void Rollback();
```

Implements

IDbTransaction.Rollback

See Also

[MySQLTransaction Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.10.2. `MySQLConnection.BeginTransaction Method (IsolationLevel)`

Syntax: Visual Basic

```
Overloads Public Function BeginTransaction( _  
    ByVal iso As IsolationLevel _  
) As MySQLTransaction
```

Syntax: C#

```
public MySQLTransaction BeginTransaction(  
IsolationLevel iso  
);
```

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLConnection.BeginTransaction Overload List](#)

23.2.4.1.2.1.1.3.1.1.11. `MySQLConnection.ChangeDatabase Method`

Syntax: Visual Basic

```
NotOverridable Public Sub ChangeDatabase( _  
    ByVal databaseName As String _  
) _  
- Implements IDbConnection.ChangeDatabase
```

Syntax: C#

```
public void ChangeDatabase(  
string databaseName  
);
```

Implements

`IDbConnection.ChangeDatabase`

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.12. MySqlConnection.Close Method

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _  
    Implements IDbConnection.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDbConnection.Close

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.13. MySqlConnection.CreateCommand Method

Syntax: Visual Basic

```
Public Function CreateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand CreateCommand();
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.14. MySqlConnection.Open Method

Syntax: Visual Basic

```
NotOverridable Public Sub Open() _  
    _
```

Implements IDbConnection.Open

Syntax: C#

```
public void Open();
```

Implements

IDbConnection.Open

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.15. MySqlConnection.Ping Method

Ping

Syntax: Visual Basic

```
Public Function Ping() As Boolean
```

Syntax: C#

```
public bool Ping();
```

Return Value

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16. MySqlConnection.InfoMessage Event

Syntax: Visual Basic

```
Public Event InfoMessage As MySqlInfoMessageEventHandler
```

Syntax: C#

```
public event MySqlInfoMessageEventHandler InfoMessage;
```

See Also

[MySQLConnection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1. [MySQLInfoMessageEventHandler Delegate](#)

Represents the method that will handle the [InfoMessage](#) event of a [MySQLConnection](#) .

Syntax: Visual Basic

```
Public Delegate Sub MySQLInfoMessageEventHandler( _  
    ByVal sender As Object, _  
    ByVal args As MySQLInfoMessageEventArgs _  
)
```

Syntax: C#

```
public delegate void MySQLInfoMessageEventHandler(  
    object sender,  
    MySQLInfoMessageEventArgs args  
);
```

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1. [MySQLInfoMessageEventArgs Class](#)

Provides data for the InfoMessage event. This class cannot be inherited.

For a list of all members of this type, see [MySQLInfoMessageEventArgs Members](#) .

Syntax: Visual Basic

Public Class MySqlInfoMessageEventArgs_
Inherits EventArgs

Syntax: C#

```
public class MySqlInfoMessageEventArgs : EventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlInfoMessageEventArgs Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1.1. MySqlInfoMessageEventArgs Members

[MySqlInfoMessageEventArgs overview](#)

Public Instance Constructors

MySqlInfoMessageEventArgs Constructor	Initializes a new instance of the MySqlInfoMessageEventArgs class.
---	--

Public Instance Fields

[errors](#)

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited)	Serves as a hash function for a particular type.

from Object)	GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
ToString(inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize(inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySQLInfoMessageEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1.1.1. MySQLInfoMessageEventArgs Constructor

Initializes a new instance of the [MySQLInfoMessageEventArgs](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySQLInfoMessageEventArgs();
```

See Also

[MySQLInfoMessageEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1.1.2. MySQLInfoMessageEventArgs.errors Field

Syntax: Visual Basic

Public errors As MySqlConnection()

Syntax: C#

```
public MySqlConnection[] errors;
```

See Also

[MySqlConnectionEventArgs Class](#) , [MySqlConnection Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1. MySqlConnection Class

Collection of error codes that can be returned by the server

For a list of all members of this type, see [MySqlConnection Members](#) .

Syntax: Visual Basic

```
Public Class MySqlConnection
```

Syntax: C#

```
public class MySqlConnection
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySqlConnection](#)

Assembly: MySqlConnection (in MySqlConnection.dll)

See Also

[MySqlConnection Members](#) , [MySqlConnection Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1. MySqlConnection Members

[MySqlError overview](#)

Public Instance Constructors

[MySqlError Constructor](#)

Public Instance Properties

Code	Error code
Level	Error level
Message	Error message

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object .
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object .

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object .

See Also

[MySqlError Class](#) , [MySql.Data.MySqlClient Namespace](#)

Syntax: Visual Basic

```
Public Sub New( _  
    ByVal level As String, _  
    ByVal code As Integer, _  
    ByVal message As String _  
)
```

Syntax: C#

```
public MySQLError(  
    stringlevel,  
    intcode,  
    stringmessage  
);
```

Parameters

- level:
- code:
- message:

See Also

[MySQLError Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1.1.2. Code Property

Error code

Syntax: Visual Basic

```
Public ReadOnly Property Code As Integer
```

Syntax: C#

```
public int Code {get;}
```

See Also

[MySQLError Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1.1.3. Level Property

Error level

Syntax: Visual Basic

```
Public ReadOnly Property Level As String
```

Syntax: C#

```
public string Level {get;}
```

See Also

[MySqlError Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.16.1.1.1.2.1.1.4. Message Property

Error message

Syntax: Visual Basic

```
Public ReadOnly Property Message As String
```

Syntax: C#

```
public string Message {get;}
```

See Also

[MySqlError Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.3.1.1.17. MySqlConnection.StateChange Event

Syntax: Visual Basic

```
Public Event StateChange As StateChangeEventHandler
```

Syntax: C#

```
public event StateChangeEventHandler StateChange;
```

See Also

[MySqlConnection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.1.4. MySqlCommand Constructor (String, MySqlConnection, MySqlTransaction)

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection, _  
    ByVal transaction As MySqlTransaction _  
)
```

Syntax: C#

```
public MySqlCommand(  
    stringcmdText,  
    MySqlConnectionconnection,  
    MySqlTransactiontransaction  
);
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
 [MySqlCommand Constructor Overload List](#)

23.2.4.1.2.1.2. CommandText Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandText As String _  
— Implements IDbCommand.CommandText
```

Syntax: C#

```
public string CommandText {get; set;}
```

Implements

IDbCommand.CommandText

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.3. CommandTimeout Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandTimeout As Integer _
```

```
— Implements IDbCommand.CommandTimeout
```

Syntax: C#

```
public int CommandTimeout {get; set;}
```

Implements

IDbCommand.CommandTimeout

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.4. CommandType Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandType As CommandType _
```

```
— Implements IDbCommand.CommandType
```

Syntax: C#

```
public System.Data.CommandType CommandType {get; set;}
```

Implements

IDbCommand.CommandType

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.5. Connection Property

Syntax: Visual Basic

```
Public Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get; set;}
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.6. IsPrepared Property

Syntax: Visual Basic

```
Public ReadOnly Property IsPrepared As Boolean
```

Syntax: C#

```
public bool IsPrepared {get;}
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7. Parameters Property

Syntax: Visual Basic

```
Public ReadOnly Property Parameters As MySqlParameterCollection
```

Syntax: C#

```
public MySqlParameterCollection Parameters {get;}
```

See Also

[MySqlCommand Class , MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1. MySqlCommandCollection Class

Represents a collection of parameters relevant to a [MySqlCommand](#) as well as their respective mappings to columns in a DataSet. This class cannot be inherited.

For a list of all members of this type, see [MySqlCommandCollection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommandCollection_  
    Inherits MarshalByRefObject_  
    Implements IDataParameterCollection, IList, ICollection, IEnumerable
```

Syntax: C#

```
public sealed class MySqlCommandCollection : MarshalByRefObject, I
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlCommandCollection Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1. MySqlCommandCollection Members

[MySqlCommandCollection overview](#)

Public Instance Constructors

MySQLParameterCollection Constructor	Initializes a new instance of the MySQLParameterCollection class.
--	---

Public Instance Properties

Count	Gets the number of MySQLParameter objects in the collection.
Item	Overloaded. Gets the MySQLParameter with a specified attribute. In C#, this property is the indexer for the MySQLParameterCollection class.

Public Instance Methods

Add	Overloaded. Adds the specified MySQLParameter object to the MySQLParameterCollection .
Clear	Removes all items from the collection.
Contains	Overloaded. Gets a value indicating whether a MySQLParameter exists in the collection.
CopyTo	Copies MySQLParameter objects from the MySQLParameterCollection to the specified array.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.

IndexOf	Overloaded. Gets the location of a MySQLParameter in the collection.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Insert	Inserts a MySQLParameter into the collection at the specified index.
Remove	Removes the specified MySQLParameter from the collection.
RemoveAt	Overloaded. Removes the specified MySQLParameter from the collection.
ToString(inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.1. MySQLParameterCollection Constructor

Initializes a new instance of the [MySQLParameterCollection](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySQLParameterCollection();
```

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.2. Count Property

Gets the number of MySQLParameter objects in the collection.

Syntax: Visual Basic

NotOverridable Public ReadOnly Property Count As Integer _

— Implements ICollection.Count

Syntax: C#

```
public int Count {get;}
```

Implements

ICollection.Count

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3. Item Property

Gets the [MySQLParameter](#) with a specified attribute. In C#, this property is the indexer for the [MySQLParameterCollection](#) class.

Overload List

Gets the [MySQLParameter](#) at the specified index.

- [public MySQLParameter this\[int\] {get; set;}](#)

Gets the [MySQLParameter](#) with the specified name.

- [public MySQLParameter this\[string\] {get; set;}](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1. MySQLParameter Class

Represents a parameter to a [MySQLCommand](#) , and optionally, its mapping to DataSet columns. This class cannot be inherited.

For a list of all members of this type, see [MySQLParameter Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameter_  
    Inherits MarshalByRefObject_  
    Implements IDataParameter, IDbDataParameter, ICloneable
```

Syntax: C#

```
public sealed class MySqlParameter : MarshalByRefObject, IDataParame
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlParameter Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.1. [MySqlParameter Members](#)

[MySqlParameter overview](#)

Public Instance Constructors

MySqlParameter	Overloaded. Initializes a new instance of the MySqlParameter class.
--------------------------------	---

Public Instance Properties

DbType	Gets or sets the DbType of the parameter.
	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return

Direction	value parameter. As of MySql version 4.1 and earlier, input-only is the only valid choice.
IsNullable	Gets or sets a value indicating whether the parameter accepts null values.
IsUnsigned	
MySqlDbType	Gets or sets the MySqlDbType of the parameter.
ParameterName	Gets or sets the name of the MySqlParameter.
Precision	Gets or sets the maximum number of digits used to represent the Value property.
Scale	Gets or sets the number of decimal places to which Value is resolved.
Size	Gets or sets the maximum size, in bytes, of the data within the column.
SourceColumn	Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the Value .
SourceVersion	Gets or sets the DataRowVersion to use when loading Value .
Value	Gets or sets the value of the parameter.

Public Instance Methods

CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.

InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ToString	Overridden. Gets a string containing the ParameterName .

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.1. MySQLParameter Constructor

Initializes a new instance of the MySQLParameter class.

Overload List

Initializes a new instance of the MySQLParameter class.

- [public MySQLParameter\(\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and the data type.

- [public MySQLParameter\(string,MySQLDbType\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySQLDbType](#) , and the size.

- [public MySQLParameter\(string,MySQLDbType,int\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersion to use, and the value of the parameter.

- [public MySQLParameter\(string,MySQLDbType,int,ParameterDirection,bool,byte,byte\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySQLDbType](#) , the size, and the source column name.

- [public MySqlConnection\(string, MySqlDbType, int, string\);](#)

Initializes a new instance of the [MySqlParameter](#) class with the parameter name and a value of the new MySqlConnection.

- [public MySqlConnection\(string, object\);](#)

See Also

[MySqlParameter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.1.1. MySqlConnection Constructor ()

Initializes a new instance of the MySqlConnection class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlConnection();
```

See Also

[MySqlParameter Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlParameter Constructor Overload List](#)

23.2.4.1.2.1.7.1.1.3.1.1.1.2. MySqlConnection Constructor (String, MySqlDbType)

Initializes a new instance of the [MySqlParameter](#) class with the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType _  
)
```

Syntax: C#

```
public MySqlParameter(
    string parameterName,
    MySqlDbType dbType
);
```

Parameters

- parameterName: The name of the parameter to map.
- dbType: One of the [MySqlDbType](#) values.

See Also

[MySqlParameter Class](#) , [MySql.Data.MySqlClient Namespace](#) , [MySqlParameter Constructor Overload List](#)

23.2.4.1.2.1.7.1.1.3.1.1.1.2.1. MySqlDbType Enumeration

Specifies MySQL specific data type of a field, property, for use in a [MySqlParameter](#) .

Syntax: Visual Basic

```
Public Enum MySqlDbType
```

Syntax: C#

```
public enum MySqlDbType
```

Members

Member Name	Description
VarChar	A variable-length string containing 0 to 65535 characters
Timestamp	A timestamp. The range is '1970-01-01 00:00:00' to sometime in the year 2037
LongBlob	A BLOB or TEXT column with a maximum length of 4294967295 or 4G (2 ³² - 1) characters
	Time

Time	The range is '-838:59:59' to '838:59:59'.
TinyBlob	A BLOB or TEXT column with a maximum length of 255 (2 ⁸ - 1) characters
Datetime	DateTime The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
Decimal	Decimal A fixed precision and scale numeric value between -10 ³⁸ -1 and 10 ³⁸ -1.
UByte	
Blob	A BLOB or TEXT column with a maximum length of 65535 (2 ¹⁶ - 1) characters
Double	Double A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.
Newdate	Obsolete Use Datetime or Date type
Byte	Byte The signed range is -128 to 127. The unsigned range is 0 to 255.
Date	Date The supported range is '1000-01-01' to '9999-12-31'.
VarChar	A variable-length string containing 0 to 255 characters
UInt16	
UInt24	
Int16	Int16 A 16-bit signed integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535

NewDecimal	New Decimal
Set	A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', ... A SET can have a maximum of 64 members.
String	Obsolete Use VarChar type
Enum	An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', ..., NULL or the special "" error value. An ENUM can have a maximum of 65535 distinct values
Geometry	
UInt64	
Int64	Int64 A 64-bit signed integer.
UInt32	
Int24	Specifies a 24 (3 byte) signed or unsigned value.
Bit	Bit-field data type
Float	Single A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.
Year	A year in 2- or 4-digit format (default is 4-digit). The allowable values are 1901 to 2155, 0000 in the 4-digit year format, and 1970-2069 if you use the 2-digit format (70-69)
Int32	Int32 A 32-bit signed integer
MediumBlob	A BLOB or TEXT column with a maximum length of 16777215 (2 ²⁴ - 1) characters

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.3.3. [MySqlParameter Constructor \(String, MySqlDbType, Int32\)](#)

Initializes a new instance of the [MySqlParameter](#) class with the parameter name, the [MySqlDbType](#) , and the size.

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType, _  
    ByVal size As Integer _  
)
```

Syntax: C#

```
public MySqlParameter(  
    stringparameterName,  
    MySqlDbTypeDbType,  
    intsize  
);
```

Parameters

- `parameterName`: The name of the parameter to map.
- `dbType`: One of the [MySqlDbType](#) values.
- `size`: The length of the parameter.

See Also

[MySqlParameter Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlParameter Constructor Overload List](#)

23.2.4.1.2.1.7.1.1.3.1.1.4. [MySqlParameter Constructor \(String, MySqlDbType, Int32, ParameterDirection, Boolean, Byte,](#)

Byte, String, DataRowVersion, Object)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersion to use, and the value of the parameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType, _  
    ByVal size As Integer, _  
    ByVal direction As ParameterDirection, _  
    ByVal isNullable As Boolean, _  
    ByVal precision As Byte, _  
    ByVal scale As Byte, _  
    ByVal sourceColumn As String, _  
    ByVal sourceVersion As DataRowVersion, _  
    ByVal value As Object _  
)
```

Syntax: C#

```
public MySQLParameter(  
    string parameterName,  
    MySqlDbType dbType,  
    int size,  
    ParameterDirection direction,  
    bool isNullable,  
    byte precision,  
    byte scale,  
    string sourceColumn,  
    DataRowVersion sourceVersion,  
    object value  
);
```

Parameters

- parameterName: The name of the parameter to map.
- dbType: One of the [MySQLDbType](#) values.
- size: The length of the parameter.

- direction: One of the ParameterDirection values.
- isNullable: true if the value of the field can be null, otherwise false.
- precision: The total number of digits to the left and right of the decimal point to which [Value](#) is resolved.
- scale: The total number of decimal places to which [Value](#) is resolved.
- sourceColumn: The name of the source column.
- sourceVersion: One of the DataRowVersion values.
- value: An Object that is the value of the [MySQLParameter](#) .

Exceptions

Exception Type	Condition
ArgumentException	

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySQLParameter Constructor Overload List](#)

23.2.4.1.2.1.7.1.1.3.1.1.1.4.1. Value Property

Gets or sets the value of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property Value As Object _
    Implements IDataParameter.Value
```

Syntax: C#

```
public object Value {get; set;}
```

Implements

IDataParameter.Value

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.1.5. MySQLParameter Constructor (String, MySqlDbType, Int32, String)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySqlDbType](#) , the size, and the source column name.

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType, _  
    ByVal size As Integer, _  
    ByVal sourceColumn As String _  
)
```

Syntax: C#

```
public MySQLParameter(  
    stringparameterName,  
    MySqlDbTypedbType,  
    intsize,  
    stringsourceColumn  
);
```

Parameters

- parameterName: The name of the parameter to map.
- dbType: One of the [MySqlDbType](#) values.
- size: The length of the parameter.
- sourceColumn: The name of the source column.

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameter Constructor Overload List](#)

23.2.4.1.2.1.7.1.1.3.1.1.1.6. MySqlCommandParameter Constructor (String, Object)

Initializes a new instance of the [MySqlCommandParameter](#) class with the parameter name and a value of the new MySqlCommandParameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal parameterName As String, _  
    ByVal value As Object _  
)
```

Syntax: C#

```
public MySqlCommandParameter(  
    stringparameterName,  
    objectvalue  
);
```

Parameters

- parameterName: The name of the parameter to map.
- value: An Object that is the value of the [MySqlCommandParameter](#).

See Also

[MySqlCommandParameter Class](#), [MySqlCommandClient Namespace](#), [MySqlCommandParameter Constructor Overload List](#)

23.2.4.1.2.1.7.1.1.3.1.1.2. DbType Property

Gets or sets the DbType of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property DbType As DbType _  
    Implements IDataParameter.DbType
```

Syntax: C#

```
public System.Data.DbType DbType {get; set;}
```

Implements

IDataParameter.DbType

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.3. Direction Property

Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySQL version 4.1 and earlier, input-only is the only valid choice.

Syntax: Visual Basic

```
NotOverridable Public Property Direction As ParameterDirection _  
    Implements IDataParameter.Direction
```

Syntax: C#

```
public System.Data.ParameterDirection Direction {get; set;}
```

Implements

IDataParameter.Direction

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.4. IsNullable Property

Gets or sets a value indicating whether the parameter accepts null values.

Syntax: Visual Basic

```
NotOverridable Public Property IsNullable As Boolean _  
    Implements IDataParameter.IsNullable
```

Syntax: C#

```
public bool IsNullable {get; set;}
```

Implements

IDataParameter.IsNullable

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.5. **IsUnsigned Property**

Syntax: Visual Basic

```
Public Property IsUnsigned As Boolean
```

Syntax: C#

```
public bool IsUnsigned {get; set;}
```

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.6. **MySQLDbType Property**

Gets or sets the MySQLDbType of the parameter.

Syntax: Visual Basic

```
Public Property MySQLDbType As MySQLDbType
```

Syntax: C#

```
public MySQLDbType MySQLDbType {get; set;}
```

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.7. ParameterName Property

Gets or sets the name of the MySqlCommandParameter.

Syntax: Visual Basic

```
NotOverridable Public Property ParameterName As String _  
    Implements IDataParameter.ParameterName
```

Syntax: C#

```
public string ParameterName {get; set;}
```

Implements

IDataParameter.ParameterName

See Also

[MySqlCommandParameter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.8. Precision Property

Gets or sets the maximum number of digits used to represent the [Value](#) property.

Syntax: Visual Basic

```
NotOverridable Public Property Precision As Byte _  
    Implements IDbDataParameter.Precision
```

Syntax: C#

```
public byte Precision {get; set;}
```

Implements

IDbDataParameter.Precision

See Also

[MySqlCommandParameter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.9. Scale Property

Gets or sets the number of decimal places to which [Value](#) is resolved.

Syntax: Visual Basic

```
NotOverridable Public Property Scale As Byte _  
    Implements IDbDataParameter.Scale
```

Syntax: C#

```
public byte Scale {get; set;}
```

Implements

IDbDataParameter.Scale

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.10. Size Property

Gets or sets the maximum size, in bytes, of the data within the column.

Syntax: Visual Basic

```
NotOverridable Public Property Size As Integer _  
    Implements IDbDataParameter.Size
```

Syntax: C#

```
public int Size {get; set;}
```

Implements

IDbDataParameter.Size

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.11. SourceColumn Property

Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the [Value](#) .

Syntax: Visual Basic

```
NotOverridable Public Property SourceColumn As String _  
    Implements IDataParameter.SourceColumn
```

Syntax: C#

```
public string SourceColumn {get; set;}
```

Implements

IDataParameter.SourceColumn

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.12. SourceVersion Property

Gets or sets the DataRowVersion to use when loading [Value](#) .

Syntax: Visual Basic

```
NotOverridable Public Property SourceVersion As DataRowVersion _  
    Implements IDataParameter.SourceVersion
```

Syntax: C#

```
public System.Data.DataRowVersion SourceVersion {get; set;}
```

Implements

IDataParameter.SourceVersion

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.1.1.13. MySQLParameter.ToString Method

Overridden. Gets a string containing the [ParameterName](#) .

Syntax: Visual Basic

Overrides Public Function ToString() As String

Syntax: C#

```
public override string ToString();
```

Return Value

See Also

[MySQLParameter Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.3.2. Item Property (Int32)

Gets the [MySQLParameter](#) at the specified index.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _  
    ByVal index As Integer _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter this[  
    int index  
] {get; set;}
```

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.Item Overload List](#)

23.2.4.1.2.1.7.1.1.3.3. Item Property (String)

Gets the [MySQLParameter](#) with the specified name.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _  
    ByVal name As String _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter this[  
    stringname  
] {get; set;}
```

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.Item Overload List](#)

23.2.4.1.2.1.7.1.1.4. Add Method

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

Overload List

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

- [public MySQLParameter Add\(MySQLParameter\);](#)

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

- [public int Add\(object\);](#)

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the parameter name and the data type.

- [public MySQLParameter Add\(string,MySQLDbType\);](#)

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) with the parameter name, the data type, and the column length.

- [public MySQLParameter Add\(string,MySQLDbType,int\);](#)

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) with the parameter name, the data type, the column length, and the source column name.

- [public MySQLParameter Add\(string,MySQLDbType,int,string\);](#)

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the specified parameter name and value.

- [public MySQLParameter Add\(string,object\);](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.4.1. [MySQLParameterCollection.Add Method \(MySQLParameter\)](#)

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#) .

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal value As MySQLParameter _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(  
MySQLParametervalue  
);
```

Parameters

- value: The [MySQLParameter](#) to add to the collection.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.Add Overload List](#)

23.2.4.1.2.1.7.1.1.4.2. MySqlConnection.Add Method (Object)

Adds the specified [MySqlParameter](#) object to the [MySqlParameterCollection](#) .

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Add( _  
    ByVal value As Object _  
) As Integer _  
- Implements IList.Add
```

Syntax: C#

```
public int Add(  
objectvalue  
);
```

Parameters

- value: The [MySqlParameter](#) to add to the collection.

Return Value

The index of the new [MySqlParameter](#) object.

Implements

IList.Add

See Also

[MySqlParameterCollection Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlParameterCollection.Add Overload List](#)

23.2.4.1.2.1.7.1.1.4.3. MySqlConnection.Add Method (String, MySqlDbType)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) given the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Function Add( _
```

```
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType _  
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(  
stringparameterName,  
MySqlDbTypedbType  
);
```

Parameters

- parameterName: The name of the parameter.
- dbType: One of the [MySqlDbType](#) values.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlParameterCollection.Add Overload List](#)

23.2.4.1.2.1.7.1.1.4.4. [MySqlParameterCollection.Add Method \(String, MySqlDbType, Int32\)](#)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, and the column length.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType, _  
    ByVal size As Integer _  
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(  
stringparameterName,  
MySqlDbTypedbType,
```

```
intsize  
);
```

Parameters

- `parameterName`: The name of the parameter.
- `dbType`: One of the [MySQLDbType](#) values.
- `size`: The length of the column.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.Add Overload List](#)

23.2.4.1.2.1.7.1.1.4.5. `MySQLParameterCollection.Add Method (String, MySQLDbType, Int32, String)`

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) with the parameter name, the data type, the column length, and the source column name.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySQLDbType, _  
    ByVal size As Integer, _  
    ByVal sourceColumn As String _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(  
stringparameterName,  
MySQLDbTypedbType,  
intsize,  
stringsourceColumn  
);
```

Parameters

- `parameterName`: The name of the parameter.
- `dbType`: One of the [MySQLDbType](#) values.
- `size`: The length of the column.
- `sourceColumn`: The name of the source column.

Return Value

The newly added [MySQLParameter](#) object.

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.Add Overload List](#)

23.2.4.1.2.1.7.1.1.4.6. `MySQLParameterCollection.Add Method (String, Object)`

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the specified parameter name and value.

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal parameterName As String, _  
    ByVal value As Object _  
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(  
stringparameterName,  
objectvalue  
);
```

Parameters

- `parameterName`: The name of the parameter.

- value: The [Value](#) of the [MySqlParameter](#) to add to the collection.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlParameterCollection.Add Overload List](#)

23.2.4.1.2.1.7.1.1.5. MySqlConnection.Clear Method

Removes all items from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Clear() _  
    Implements IList.Clear
```

Syntax: C#

```
public void Clear();
```

Implements

IList.Clear

See Also

[MySqlParameterCollection Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.6. Contains Method

Gets a value indicating whether a [MySqlParameter](#) exists in the collection.

Overload List

Gets a value indicating whether a [MySqlParameter](#) exists in the collection.

- [public bool Contains\(object\);](#)

Gets a value indicating whether a [MySQLParameter](#) with the specified parameter name exists in the collection.

- [public bool Contains\(string\);](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.6.1. [MySQLParameterCollection.Contains Method \(Object\)](#)

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _  
    ByVal value As Object _  
) As Boolean _  
- Implements IList.Contains
```

Syntax: C#

```
public bool Contains(  
objectvalue  
);
```

Parameters

- value: The value of the [MySQLParameter](#) object to find.

Return Value

true if the collection contains the [MySQLParameter](#) object; otherwise, false.

Implements

[IList.Contains](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,

[MySQLParameterCollection.Contains Overload List](#)

23.2.4.1.2.1.7.1.1.6.2. MySQLParameterCollection.Contains Method (String)

Gets a value indicating whether a [MySQLParameter](#) with the specified parameter name exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _  
    ByVal name As String _  
) As Boolean _  
- Implements IDataParameterCollection.Contains
```

Syntax: C#

```
public bool Contains(  
stringname  
);
```

Parameters

- name: The name of the [MySQLParameter](#) object to find.

Return Value

true if the collection contains the parameter; otherwise, false.

Implements

IDataParameterCollection.Contains

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.Contains Overload List](#)

23.2.4.1.2.1.7.1.1.7. MySQLParameterCollection.CopyTo Method

Copies MySQLParameter objects from the MySQLParameterCollection to the specified array.

Syntax: Visual Basic

```
NotOverridable Public Sub CopyTo( _  
    ByVal array As Array, _  
    ByVal index As Integer _  
) _  
- Implements ICollection.CopyTo
```

Syntax: C#

```
public void CopyTo(  
Arrayarray,  
intindex  
);
```

Parameters

- array:
- index:

Implements

ICollection.CopyTo

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.8. IndexOf Method

Gets the location of a [MySQLParameter](#) in the collection.

Overload List

Gets the location of a [MySQLParameter](#) in the collection.

- [public int IndexOf\(object\);](#)

Gets the location of the [MySQLParameter](#) in the collection with a specific parameter name.

- [public int IndexOf\(string\);](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.8.1. MySQLParameterCollection.IndexOf Method (Object)

Gets the location of a [MySQLParameter](#) in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _  
    ByVal value As Object _  
) As Integer _  
— Implements IList.IndexOf
```

Syntax: C#

```
public int IndexOf(  
objectvalue  
);
```

Parameters

- value: The [MySQLParameter](#) object to locate.

Return Value

The zero-based location of the [MySQLParameter](#) in the collection.

Implements

IList.IndexOf

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.IndexOf Overload List](#)

23.2.4.1.2.1.7.1.1.8.2. MySQLParameterCollection.IndexOf Method (String)

Gets the location of the [MySQLParameter](#) in the collection with a specific parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _  
    ByVal parameterName As String _  
) As Integer _  
— Implements IDataParameterCollection.IndexOf
```

Syntax: C#

```
public int IndexOf(  
stringparameterName  
);
```

Parameters

- parameterName: The name of the [MySQLParameter](#) object to retrieve.

Return Value

The zero-based location of the [MySQLParameter](#) in the collection.

Implements

[IDataParameterCollection.IndexOf](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.IndexOf Overload List](#)

23.2.4.1.2.1.7.1.1.9. MySQLParameterCollection.Insert Method

Inserts a MySQLParameter into the collection at the specified index.

Syntax: Visual Basic

```
NotOverridable Public Sub Insert( _  
    ByVal index As Integer, _  
    ByVal value As Object _
```

```
) _  
- Implements IList.Insert
```

Syntax: C#

```
public void Insert(  
int index,  
object value  
);
```

Parameters

- index:
- value:

Implements

IList.Insert

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.10. MySQLParameterCollection.Remove Method

Removes the specified MySQLParameter from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Remove( _  
    ByVal value As Object _  
) _  
- Implements IList.Remove
```

Syntax: C#

```
public void Remove(  
object value  
);
```

Parameters

- value:

Implements

ICollection.Remove

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.11. RemoveAt Method

Removes the specified [MySQLParameter](#) from the collection.

Overload List

Removes the specified [MySQLParameter](#) from the collection using a specific index.

- [public void RemoveAt\(int\);](#)

Removes the specified [MySQLParameter](#) from the collection using the parameter name.

- [public void RemoveAt\(string\);](#)

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.7.1.1.11.1. MySQLParameterCollection.RemoveAt Method (Int32)

Removes the specified [MySQLParameter](#) from the collection using a specific index.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _  
    ByVal index As Integer _  
) _  
_
```

Implements `IList.RemoveAt`

Syntax: C#

```
public void RemoveAt(  
int index  
);
```

Parameters

- index: The zero-based index of the parameter.

Implements

`IList.RemoveAt`

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.RemoveAt Overload List](#)

23.2.4.1.2.1.7.1.1.11.2. `MySQLParameterCollection.RemoveAt Method (String)`

Removes the specified [MySQLParameter](#) from the collection using the parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _  
    ByVal name As String _  
) _  
— Implements IDataParameterCollection.RemoveAt
```

Syntax: C#

```
public void RemoveAt(  
string name  
);
```

Parameters

- name: The name of the [MySQLParameter](#) object to retrieve.

Implements

`IDataParameterCollection.RemoveAt`

See Also

[MySQLParameterCollection Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLParameterCollection.RemoveAt Overload List](#)

23.2.4.1.2.1.8. Transaction Property

Syntax: Visual Basic

Public Property Transaction As MySqlTransaction

Syntax: C#

```
public MySqlTransaction Transaction {get; set;}
```

See Also

[MySQLCommand Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.9. UpdatedRowSource Property

Syntax: Visual Basic

NotOverridable Public Property UpdatedRowSource As UpdateRowSource _
_ Implements IDbCommand.UpdatedRowSource

Syntax: C#

```
public System.Data.UpdateRowSource UpdatedRowSource {get; set;}
```

Implements

`IDbCommand.UpdatedRowSource`

See Also

[MySQLCommand Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.10. MySqlCommand.Cancel Method

Attempts to cancel the execution of a MySqlCommand. This operation is not supported.

Syntax: Visual Basic

```
NotOverridable Public Sub Cancel() _  
    Implements IDbCommand.Cancel
```

Syntax: C#

```
public void Cancel();
```

Implements

IDbCommand.Cancel

Remarks

Cancelling an executing command is currently not supported on any version of MySQL.

Exceptions

Exception Type	Condition
NotSupportedException	This operation is not supported.

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.11. MySqlCommand.CreateParameter Method

Creates a new instance of a [MySqlParameter](#) object.

Syntax: Visual Basic

```
Public Function CreateParameter() As SqlParameter
```

Syntax: C#

```
public MySqlParameter CreateParameter();
```

Return Value

A [MySqlParameter](#) object.

Remarks

This method is a strongly-typed version of CreateParameter.

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.12. MySqlCommand.ExecuteNonQuery Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteNonQuery() As Integer _  
    Implements IDbCommand.ExecuteNonQuery
```

Syntax: C#

```
public int ExecuteNonQuery();
```

Implements

IDbCommand.ExecuteNonQuery

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13. ExecuteReader Method

Overload List

- [public MySqlDataReader ExecuteReader\(\);](#)

- [public MySqlCommand.ExecuteReader\(CommandBehavior\);](#)

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1. MySqlCommand.ExecuteReader Method ()

Syntax: Visual Basic

Overloads Public Function ExecuteReader() As MySqlDataReader

Syntax: C#

```
public MySqlDataReader ExecuteReader();
```

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlCommand.ExecuteReader Overload List](#)

23.2.4.1.2.1.13.1.1. MySqlDataReader Class

Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.

For a list of all members of this type, see [MySqlDataReader Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataReader_  
    Inherits MarshalByRefObject_  
    Implements IEnumerable, IDataReader, IDisposable, IDataRecord
```

Syntax: C#

```
public sealed class MySqlDataReader : MarshalByRefObject, IEnumerable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-

safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDataReader Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1. MySqlDataReader Members

[MySqlDataReader overview](#)

Public Instance Properties

Depth	Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.
FieldCount	Gets the number of columns in the current row.
HasRows	Gets a value indicating whether the MySqlDataReader contains one or more rows.
IsClosed	Gets a value indicating whether the data reader is closed.
Item	Overloaded. Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Public Instance Methods

Close	Closes the MySqlDataReader object.
CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBoolean	Gets the value of the specified column as a Boolean.
GetByte	Gets the value of the specified column as a byte.
GetBytes	Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.
GetChar	Gets the value of the specified column as a single character.
GetChars	Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.
GetDataTypeName	Gets the name of the source data type.
GetDateTime	
GetDecimal	
GetDouble	
GetFieldType	Gets the Type that is the data type of the object.
GetFloat	
GetGuid	
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetInt16	
GetInt32	
GetInt64	
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetMySqlDateTime	
GetName	Gets the name of the specified column.

GetOrdinal	Gets the column ordinal, given the name of the column.
GetSchemaTable	Returns a DataTable that describes the column metadata of the MySqlDataReader.
GetString	
GetTimeSpan	
GetType(inherited from Object)	Gets the Typeof the current instance.
GetUInt16	
GetUInt32	
GetUInt64	
GetValue	Gets the value of the specified column in its native format.
GetValues	Gets all attribute columns in the collection for the current row.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
IsDBNull	Gets a value indicating whether the column contains non-existent or missing values.
NextResult	Advances the data reader to the next result, when reading the results of batch SQL statements.
Read	Advances the MySqlDataReader to the next record.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.1. Depth Property

Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.

Syntax: Visual Basic

NotOverridable Public ReadOnly Property Depth As Integer _

— Implements IDataReader.Depth

Syntax: C#

```
public int Depth {get;}
```

Implements

IDataReader.Depth

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.2. FieldCount Property

Gets the number of columns in the current row.

Syntax: Visual Basic

NotOverridable Public ReadOnly Property FieldCount As Integer _

— Implements IDataRecord.FieldCount

Syntax: C#

```
public int FieldCount {get;}
```

Implements

IDataRecord.FieldCount

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.3. HasRows Property

Gets a value indicating whether the `MySqlDataReader` contains one or more rows.

Syntax: Visual Basic

```
Public ReadOnly Property HasRows As Boolean
```

Syntax: C#

```
public bool HasRows {get;}
```

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.4. **IsClosed Property**

Gets a value indicating whether the data reader is closed.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsClosed As Boolean _  
    Implements IDataReader.IsClosed
```

Syntax: C#

```
public bool IsClosed {get;}
```

Implements

`IDataReader.IsClosed`

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.5. **Item Property**

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Overload List

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

- [public object this\[int\] {get;}](#)

Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

- [public object this\[string\] {get;}](#)

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.5.1. Item Property (Int32)

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _  
    ByVal i As Integer _  
) _  
- Implements IDataRecord.Item As Object _  
- Implements IDataRecord.Item
```

Syntax: C#

```
public object this[  
    inti  
] {get;}
```

Implements

`IDataRecord.Item`

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlDataReader.Item Overload List](#)

23.2.4.1.2.1.13.1.1.1.5.2. Item Property (String)

Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _  
    ByVal name As String _  
) _  
- Implements IDataRecord.Item As Object _  
- Implements IDataRecord.Item
```

Syntax: C#

```
public object this[  
    stringname  
] {get;}
```

Implements

IDataRecord.Item

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlDataReader.Item Overload List](#)

23.2.4.1.2.1.13.1.1.1.6. RecordsAffected Property

Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property RecordsAffected As Integer _  
- Implements IDataReader.RecordsAffected
```

Syntax: C#

```
public int RecordsAffected {get;}
```

Implements

IDataReader.RecordsAffected

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.7. [MySqlDataReader.Close Method](#)

Closes the MySqlDataReader object.

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _  
    Implements IDataReader.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDataReader.Close

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.8. [MySqlDataReader.GetBoolean Method](#)

Gets the value of the specified column as a Boolean.

Syntax: Visual Basic

```
NotOverridable Public Function GetBoolean( _  
    ByVal i As Integer _
```

```
) As Boolean _  
- Implements IDataRecord.GetBoolean
```

Syntax: C#

```
public bool GetBoolean(  
inti  
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetBoolean

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.9. MySQLDataReader.GetByte Method

Gets the value of the specified column as a byte.

Syntax: Visual Basic

```
NotOverridable Public Function GetByte( _  
    ByVal i As Integer _  
    ) As Byte _
```

```
- Implements IDataRecord.GetByte
```

Syntax: C#

```
public byte GetByte(  
inti  
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetByte

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.10. MySQLDataReader.GetBytes Method

Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetBytes( _  
    ByVal i As Integer, _  
    ByVal dataIndex As Long, _  
    ByVal buffer As Byte(), _  
    ByVal bufferIndex As Integer, _  
    ByVal length As Integer _  
) As Long _  
- Implements IDataRecord.GetBytes
```

Syntax: C#

```
public long GetBytes(  
    inti,  
    longdataIndex,  
    byte[]buffer,  
    intbufferIndex,  
    intlength  
);
```

Parameters

- i: The zero-based column ordinal.
- dataIndex: The index within the field from which to begin the read

operation.

- `buffer`: The buffer into which to read the stream of bytes.
- `bufferIndex`: The index for buffer to begin the read operation.
- `length`: The maximum length to copy into the buffer.

Return Value

The actual number of bytes read.

Implements

`IDataRecord.GetBytes`

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.11. `MySqlDataReader.GetChar` Method

Gets the value of the specified column as a single character.

Syntax: Visual Basic

```
NotOverridable Public Function GetChar( _  
    ByVal i As Integer _  
) As Char _  
— Implements IDataRecord.GetChar
```

Syntax: C#

```
public char GetChar(  
    inti  
);
```

Parameters

- `i`:

Return Value

Implements

IDataRecord.GetChar

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.12. MySQLDataReader.GetChars Method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetChars( _  
    ByVal i As Integer, _  
    ByVal fieldOffset As Long, _  
    ByVal buffer As Char(), _  
    ByVal bufferoffset As Integer, _  
    ByVal length As Integer _  
) As Long _  
-  
- Implements IDataRecord.GetChars
```

Syntax: C#

```
public long GetChars(  
    inti,  
    longfieldOffset,  
    char[]buffer,  
    intbufferoffset,  
    intlength  
);
```

Parameters

- i:
- fieldOffset:
- buffer:

- bufferoffset:
- length:

Return Value

Implements

IDataRecord.GetChars

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.13. [MySqlDataReader.GetDataTypeName Method](#)

Gets the name of the source data type.

Syntax: Visual Basic

```
NotOverridable Public Function GetDataTypeName( _  
    ByVal i As Integer _  
) As String _  
- Implements IDataRecord.GetDataTypeName
```

Syntax: C#

```
public string GetDataTypeName(  
    inti  
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetDataTypeName

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.14. **MySQLDataReader.GetDateTime Method**

Syntax: Visual Basic

```
NotOverridable Public Function GetDateTime( _  
    ByVal index As Integer _  
) As Date _  
- Implements IDataRecord.GetDateTime
```

Syntax: C#

```
public DateTime GetDateTime(  
int index  
);
```

Implements

IDataRecord.GetDateTime

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.15. **MySQLDataReader.GetDecimal Method**

Syntax: Visual Basic

```
NotOverridable Public Function GetDecimal( _  
    ByVal index As Integer _  
) As Decimal _  
- Implements IDataRecord.GetDecimal
```

Syntax: C#

```
public decimal GetDecimal(  
int index  
);
```

Implements

IDataRecord.GetDecimal

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.16. **MySqlDataReader.GetDouble Method**

Syntax: Visual Basic

```
NotOverridable Public Function GetDouble( _  
    ByVal index As Integer _  
) As Double _
```

— Implements IDataRecord.GetDouble

Syntax: C#

```
public double GetDouble(  
int index  
);
```

Implements

IDataRecord.GetDouble

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.17. **MySqlDataReader.GetFieldType Method**

Gets the Type that is the data type of the object.

Syntax: Visual Basic

```
NotOverridable Public Function GetFieldType( _  
    ByVal i As Integer _  
) As Type _
```

— Implements IDataRecord.GetFieldType

Syntax: C#

```
public Type GetFieldType(  
    int i  
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetFieldType

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.18. MySQLDataReader.GetFloat Method

Syntax: Visual Basic

```
NotOverridable Public Function GetFloat( _  
    ByVal index As Integer _  
) As Single _
```

— Implements IDataRecord.GetFloat

Syntax: C#

```
public float GetFloat(  
    int index  
);
```

Implements

IDataRecord.GetFloat

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.19. MySqlDataReader.GetGuid Method

Syntax: Visual Basic

```
NotOverridable Public Function GetGuid( _  
    ByVal index As Integer _  
) As Guid _  
- Implements IDataRecord.GetGuid
```

Syntax: C#

```
public Guid GetGuid(  
int index  
);
```

Implements

IDataRecord.GetGuid

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.20. MySqlDataReader.GetInt16 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt16( _  
    ByVal index As Integer _  
) As Short _  
- Implements IDataRecord.GetInt16
```

Syntax: C#

```
public short GetInt16(  
int index  
);
```

Implements

IDataRecord.GetInt16

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.21. [MySQLDataReader.GetInt32 Method](#)

Syntax: Visual Basic

```
NotOverridable Public Function GetInt32( _  
    ByVal index As Integer _  
) As Integer _  
- Implements IDataRecord.GetInt32
```

Syntax: C#

```
public int GetInt32(  
    int index  
);
```

Implements

[IDataRecord.GetInt32](#)

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.22. [MySQLDataReader.GetInt64 Method](#)

Syntax: Visual Basic

```
NotOverridable Public Function GetInt64( _  
    ByVal index As Integer _  
) As Long _  
- Implements IDataRecord.GetInt64
```

Syntax: C#

```
public long GetInt64(  
    int index  
);
```

Implements

IDataRecord.GetInt64

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.23. MySqlDataReader.GetMySqlDateTime Method

Syntax: Visual Basic

```
Public Function GetMySqlDateTime( _  
    ByVal index As Integer _  
) As MySqlDateTime
```

Syntax: C#

```
public MySqlDateTime GetMySqlDateTime(  
int index  
);
```

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.24. MySqlDataReader.GetName Method

Gets the name of the specified column.

Syntax: Visual Basic

```
NotOverridable Public Function GetName( _  
    ByVal i As Integer _  
) As String _
```

— Implements IDataRecord.GetName

Syntax: C#

```
public string GetName(  
inti  
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetName

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.25. MySQLDataReader.GetOrdinal Method

Gets the column ordinal, given the name of the column.

Syntax: Visual Basic

```
NotOverridable Public Function GetOrdinal( _  
    ByVal name As String _  
) As Integer _  
- Implements IDataRecord.GetOrdinal
```

Syntax: C#

```
public int GetOrdinal(  
stringname  
);
```

Parameters

- name:

Return Value

Implements

IDataRecord.GetOrdinal

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.26. [MySqlDataReader.GetSchemaTable Method](#)

Returns a DataTable that describes the column metadata of the MySqlDataReader.

Syntax: Visual Basic

```
NotOverridable Public Function GetSchemaTable() As DataTable _  
    Implements IDataReader.GetSchemaTable
```

Syntax: C#

```
public DataTable GetSchemaTable();
```

Return Value

Implements

IDataReader.GetSchemaTable

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.27. [MySqlDataReader.GetString Method](#)

Syntax: Visual Basic

```
NotOverridable Public Function GetString( _  
    ByVal index As Integer _  
) As String _  
    Implements IDataRecord.GetString
```

Syntax: C#

```
public string GetString(  
int index
```

);

Implements

IDataRecord.GetString

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.28. MySQLDataReader.GetTimeSpan Method

Syntax: Visual Basic

```
Public Function GetTimeSpan( _  
    ByVal index As Integer _  
) As TimeSpan
```

Syntax: C#

```
public TimeSpan GetTimeSpan(  
int index  
);
```

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.29. MySQLDataReader.GetUInt16 Method

Syntax: Visual Basic

```
Public Function GetUInt16( _  
    ByVal index As Integer _  
) As UInt16
```

Syntax: C#

```
public ushort GetUInt16(  
int index  
);
```

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.30. MySqlDataReader.GetUInt32 Method

Syntax: Visual Basic

```
Public Function GetUInt32( _  
    ByVal index As Integer _  
) As UInt32
```

Syntax: C#

```
public uint GetUInt32(  
intindex  
);
```

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.31. MySqlDataReader.GetUInt64 Method

Syntax: Visual Basic

```
Public Function GetUInt64( _  
    ByVal index As Integer _  
) As UInt64
```

Syntax: C#

```
public ulong GetUInt64(  
intindex  
);
```

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.32. MySqlDataReader.GetValue Method

Gets the value of the specified column in its native format.

Syntax: Visual Basic

```
NotOverridable Public Function GetValue( _  
    ByVal i As Integer _  
    ) As Object _  
- Implements IDataRecord.GetValue
```

Syntax: C#

```
public object GetValue(  
    inti  
    );
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetValue

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.33. MySQLDataReader.GetValues Method

Gets all attribute columns in the collection for the current row.

Syntax: Visual Basic

```
NotOverridable Public Function GetValues( _  
    ByVal values As Object() _  
    ) As Integer _  
- Implements IDataRecord.GetValues
```

Syntax: C#

```
public int GetValues(  
    object[]values  
    );
```

Parameters

- values:

Return Value

Implements

IDataRecord.GetValues

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.34. [MySqlDataReader.IsDBNull Method](#)

Gets a value indicating whether the column contains non-existent or missing values.

Syntax: Visual Basic

```
NotOverridable Public Function IsDBNull( _  
    ByVal i As Integer _  
) As Boolean _  
— Implements IDataRecord.IsDBNull
```

Syntax: C#

```
public bool IsDBNull(  
    inti  
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.IsDBNull

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.35. MySqlDataReader.NextResult Method

Advances the data reader to the next result, when reading the results of batch SQL statements.

Syntax: Visual Basic

```
NotOverridable Public Function NextResult() As Boolean _  
    Implements IDataReader.NextResult
```

Syntax: C#

```
public bool NextResult();
```

Return Value

Implements

IDataReader.NextResult

See Also

[MySqlDataReader Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.1.1.1.36. MySqlDataReader.Read Method

Advances the MySqlDataReader to the next record.

Syntax: Visual Basic

```
NotOverridable Public Function Read() As Boolean _  
    Implements IDataReader.Read
```

Syntax: C#

```
public bool Read();
```

Return Value

Implements

IDataReader.Read

See Also

[MySQLDataReader Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.13.2. MySqlCommand.ExecuteReader Method (CommandBehavior)

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader( _  
    ByVal behavior As CommandBehavior _  
) As MySQLDataReader
```

Syntax: C#

```
public MySQLDataReader ExecuteReader(  
    CommandBehavior behavior  
);
```

See Also

[MySQLCommand Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLCommand.ExecuteReader Overload List](#)

23.2.4.1.2.1.14. MySqlCommand.ExecuteScalar Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteScalar() As Object _  
    Implements IDbCommand.ExecuteScalar
```

Syntax: C#

```
public object ExecuteScalar();
```

Implements

IDbCommand.ExecuteScalar

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.2.1.15. MySqlCommand.Prepare Method

Syntax: Visual Basic

```
NotOverridable Public Sub Prepare() _  
    Implements IDbCommand.Prepare
```

Syntax: C#

```
public void Prepare();
```

Implements

IDbCommand.Prepare

See Also

[MySqlCommand Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3. MySqlCommandBuilder Class

For a list of all members of this type, see [MySqlCommandBuilder Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommandBuilder_  
    Inherits Component
```

Syntax: C#

```
public sealed class MySqlCommandBuilder : Component
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for

multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlCommandBuilder Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1. MySqlCommandBuilder Members

[MySqlCommandBuilder overview](#)

Public Static (Shared) Methods

DeriveParameters	Overloaded. Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.
----------------------------------	--

Public Instance Constructors

MySqlCommandBuilder	Overloaded. Initializes a new instance of the MySqlCommandBuilder class.
-------------------------------------	--

Public Instance Properties

Container(inherited from Component)	Gets the IContainer that contains the Component.
DataAdapter	
QuotePrefix	
QuoteSuffix	
Site(inherited from Component)	Gets or sets the ISite of the Component.

Public Instance Methods

CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetDeleteCommand	
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetInsertCommand	
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.
GetUpdateCommand	
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
RefreshSchema	
ToString(inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
------------------------------------	---

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.1. DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

Overload List

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

- [public static void DeriveParameters\(MySqlCommand\);](#)
- [public static void DeriveParameters\(MySqlCommand,bool\);](#)

See Also

[MySqlCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.1.1. MySqlCommandBuilder.DeriveParameters Method (MySqlCommand)

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _  
    ByVal command As MySqlCommand _  
)
```

Syntax: C#

```
public static void DeriveParameters(  
MySqlCommandcommand  
);
```

Parameters

- **command:** The MySqlCommand referencing the stored procedure from which the parameter information is to be derived. The derived parameters are added to the Parameters collection of the MySqlCommand.

Exceptions

Exception Type	Condition
InvalidOperationException	The command text is not a valid stored procedure name.

See Also

[MySqlCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySqlCommandBuilder.DeriveParameters Overload List](#)

23.2.4.1.3.1.1.2. MySqlCommandBuilder.DeriveParameters Method (MySqlCommand, Boolean)

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _
    ByVal command As MySqlCommand, _
    ByVal useProc As Boolean _
)
```

Syntax: C#

```
public static void DeriveParameters(
    MySqlCommand command,
    bool useProc
);
```

See Also

[MySqlCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) , [MySqlCommandBuilder.DeriveParameters Overload List](#)

23.2.4.1.3.1.2. MySqlCommandBuilder Constructor

Initializes a new instance of the [MySqlCommandBuilder](#) class.

Overload List

Initializes a new instance of the [MySQLCommandBuilder](#) class.

- [public MySQLCommandBuilder\(\);](#)
- [public MySQLCommandBuilder\(MySqlDataAdapter\);](#)
- [public MySQLCommandBuilder\(MySqlDataAdapter,bool\);](#)
- [public MySQLCommandBuilder\(bool\);](#)

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.1. MySQLCommandBuilder Constructor ()

Initializes a new instance of the [MySQLCommandBuilder](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySQLCommandBuilder();
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLCommandBuilder Constructor Overload List](#)

23.2.4.1.3.1.2.2. MySQLCommandBuilder Constructor (MySqlDataAdapter)

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal adapter As MySqlDataAdapter _  
)
```

Syntax: C#

```
public MySQLCommandBuilder(
```

```
MySqlDataAdapteradapter  
);
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlCommandBuilder Constructor Overload List](#)

23.2.4.1.3.1.2.2.1. MySqlConnection Class

For a list of all members of this type, see [MySqlConnection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_  
    Inherits DbDataAdapter
```

Syntax: C#

```
public sealed class MySqlConnection : DbDataAdapter
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySqlConnection (in MySqlConnection.dll)

See Also

[MySqlConnection Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1. MySqlConnection Members

[MySqlConnection overview](#)

Public Instance Constructors

MySqlDataAdapter	Overloaded. Initializes a new instance of the MySqlDataAdapter class.
----------------------------------	---

Public Instance Properties

AcceptChangesDuringFill(inherited from DataAdapter)	Gets or sets a value indicating whether AcceptChanges is called on a DataRow after it is added to the DataTable during any of the Fill operations.
AcceptChangesDuringUpdate(inherited from DataAdapter)	Gets or sets whether AcceptChanges is called during a Update.
Container(inherited from Component)	Gets the IContainer that contains the Component.
ContinueUpdateOnError(inherited from DataAdapter)	Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.
DeleteCommand	Overloaded.
FillLoadOption(inherited from DataAdapter)	Gets or sets the LoadOption that determines how the adapter fills the DataTable from the DbDataReader.
InsertCommand	Overloaded.
MissingMappingAction(inherited from DataAdapter)	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction(inherited from DataAdapter)	Determines the action to take when existing DataSet schema does not match incoming data.
ReturnProviderSpecificTypes(inherited from DataAdapter)	Gets or sets whether the Fill method should return provider-specific values or common CLS-compliant values.
SelectCommand	Overloaded.
Site(inherited from Component)	Gets or sets the ISite of the Component.
	Gets a collection that provides the

TableMappings(inherited from DataAdapter)	master mapping between a source table and a DataTable.
UpdateBatchSize(inherited from DbDataAdapter)	Gets or sets a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
UpdateCommand	Overloaded.

Public Instance Methods

CreateObjRef(inherited from MarshalByRefObject)	Creates an object that contains the relevant information required to generate a proxy used to communicate with a remote object.
Dispose(inherited from Component)	Releases all resources used by the Component.
Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
Fill(inherited from DbDataAdapter)	Overloaded. Adds or refreshes rows in the DataSet to match rows in the data source using the DataSetname, and creates a DataTable named "Table."
FillSchema(inherited from DbDataAdapter)	Overloaded. Configures the schema of the specified DataTable based on the specified SchemaType.
GetFillParameters(inherited from DbDataAdapter)	Gets the parameters set by the user when executing an SQL SELECT statement.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures.

	a hash table.
GetLifetimeService(inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType(inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService(inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ResetFillLoadOption(inherited from DataAdapter)	Resets FillLoadOption to its default state and causes Fill to honor AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill(inherited from DataAdapter)	Determines whether the AcceptChangesDuringFill property should be persisted.
ShouldSerializeFillLoadOption(inherited from DataAdapter)	Determines whether the FillLoadOption property should be persisted.
ToString(inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.
Update(inherited from DbDataAdapter)	Overloaded. Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet.

Public Instance Events

Disposed(inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
FillError(inherited from DataAdapter)	Returned when an error occurs during a fill operation.
RowUpdated	Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event

	fires.
RowUpdating	Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Protected Internal Instance Properties

FillCommandBehavior(inherited from DbDataAdapter)	Gets or sets the behavior of the command used to fill the data adapter.
---	---

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.1. MySqlDataAdapter Constructor

Initializes a new instance of the [MySqlDataAdapter](#) class.

Overload List

Initializes a new instance of the [MySqlDataAdapter](#) class.

- [public MySqlDataAdapter\(\);](#)
- [public MySqlDataAdapter\(MySqlCommand\);](#)
- [public MySqlDataAdapter\(string, MySqlConnection\);](#)
- [public MySqlDataAdapter\(string, string\);](#)

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.1. MySqlDataAdapter Constructor ()

Initializes a new instance of the [MySqlDataAdapter](#) class.

Syntax: Visual Basic

Overloads Public Sub New()

Syntax: C#

```
public MySqlDataAdapter();
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlDataAdapter Constructor Overload List](#)

23.2.4.1.3.1.2.2.1.1.1.2. **MySqlDataAdapter Constructor (MySqlCommand)**

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommand As MySqlCommand _  
)
```

Syntax: C#

```
public MySqlDataAdapter(  
    MySqlCommand selectCommand  
);
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlDataAdapter Constructor Overload List](#)

23.2.4.1.3.1.2.2.1.1.1.3. **MySqlDataAdapter Constructor (String, MySqlConnection)**

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommandText As String, _  
    ByVal connection As MySqlConnection _  
)
```

Syntax: C#

```
public MySqlDataAdapter(  
    string selectCommandText,  
    MySqlConnection connection
```

);

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlDataAdapter Constructor Overload List](#)

23.2.4.1.3.1.2.2.1.1.1.4. **MySqlDataAdapter Constructor (String, String)**

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommandText As String, _  
    ByVal selectConnString As String _  
)
```

Syntax: C#

```
public MySqlDataAdapter(  
    stringselectCommandText,  
    stringselectConnString  
);
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlDataAdapter Constructor Overload List](#)

23.2.4.1.3.1.2.2.1.1.2. **DeleteCommand Property**

Syntax: Visual Basic

```
Overloads Public Property DeleteCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand DeleteCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.3. **InsertCommand Property**

Syntax: Visual Basic

Overloads Public Property InsertCommand As MySqlCommand

Syntax: C#

```
new public MySqlCommand InsertCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.4. SelectCommand Property

Syntax: Visual Basic

Overloads Public Property SelectCommand As MySqlCommand

Syntax: C#

```
new public MySqlCommand SelectCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.5. UpdateCommand Property

Syntax: Visual Basic

Overloads Public Property UpdateCommand As MySqlCommand

Syntax: C#

```
new public MySqlCommand UpdateCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.6. MySqlDataAdapter.RowUpdated Event

Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdated As MySqlCommandUpdatedEventHandler
```

Syntax: C#

```
public event MySqlCommandUpdatedEventHandler RowUpdated;
```

Event Data

The event handler receives an argument of type [MySqlCommandUpdatedEventArgs](#) containing data related to this event. The following MySqlCommandUpdatedEventArgs properties provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand executed when Update is called.
Errors	Gets any errors generated by the .NET Framework data provider when the Command was executed.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row	Gets the DataRow sent through an Update.
RowCount	Gets the number of rows processed in a batch of updated records.
StatementType	Gets the type of SQL statement executed.
Status	Gets the UpdateStatus of the Command property.
TableMapping	Gets the DataTableMapping sent through an Update.

See Also

[MySqlDataAdapter Class](#) , [MySql.Data.MySqlClient Namespace](#)

Represents the method that will handle the RowUpdated event of a [MySqlDataAdapter](#) .

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatedEventHandler( _  
    ByVal sender As Object, _  
    ByVal e As MySqlRowUpdatedEventArgs _  
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatedEventHandler(  
    object sender,  
    MySqlRowUpdatedEventArgs  
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.6.1.1. MySqlRowUpdatedEventArgs Class

Provides data for the RowUpdated event. This class cannot be inherited.

For a list of all members of this type, see [MySqlRowUpdatedEventArgs Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatedEventArgs_  
    Inherits RowUpdatedEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatedEventArgs : RowUpdatedEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlRowUpdatedEventArgs Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.6.1.1.1. MySqlRowUpdatedEventArgs Members

[MySqlRowUpdatedEventArgs overview](#)

Public Instance Constructors

MySqlRowUpdatedEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatedEventArgs class.
--	---

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand executed when Update is called.
Errors(inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command was executed.
RecordsAffected(inherited from RowUpdatedEventArgs)	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row(inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an Update.
RowCount(inherited from RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.

StatementType(inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status(inherited from RowUpdatedEventArgs)	Gets the UpdateStatusof the Commandproperty.
TableMapping(inherited from RowUpdatedEventArgs)	Gets the DataTableMappingsent through an Update.

Public Instance Methods

CopyToRows(inherited from RowUpdatedEventArgs)	Overloaded. Copies references to the modified rows into the provided array.
Equals(inherited from Object)	Determines whether the specified Objectis equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCodeis suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Typeof the current instance.
ToString(inherited from Object)	Returns a Stringthat represents the current Object.

See Also

[MySQLRowUpdatedEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.6.1.1.1.1. MySQLRowUpdatedEventArgs Constructor

Initializes a new instance of the MySQLRowUpdatedEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
    ByVal row As DataRow, _
    ByVal command As IDbCommand, _
    ByVal statementType As StatementType, _
```

```
    ByVal tableMapping As DataTableMapping _  
)
```

Syntax: C#

```
public MySqlRowUpdatedEventArgs(  
    DataRow row,  
    IDbCommand command,  
    StatementType statementType,  
    DataTableMapping tableMapping  
);
```

Parameters

- row: The DataRow sent through an Update.
- command: The IDbCommand executed when Update is called.
- statementType: One of the StatementType values that specifies the type of query executed.
- tableMapping: The DataTableMapping sent through an Update.

See Also

[MySqlRowUpdatedEventArgs Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.6.1.1.1.2. Command Property

Gets or sets the MySqlCommand executed when Update is called.

Syntax: Visual Basic

Overloads Public ReadOnly Property Command As MySqlCommand

Syntax: C#

```
new public MySqlCommand Command {get;}
```

See Also

[MySqlRowUpdatedEventArgs Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.7. MySqlConnection.RowUpdating Event

Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdating As MySqlConnectionRowUpdatingEventHandler
```

Syntax: C#

```
public event MySqlConnectionRowUpdatingEventHandler RowUpdating;
```

Event Data

The event handler receives an argument of type [MySqlConnectionRowUpdatingEventArgs](#) containing data related to this event. The following MySqlConnectionRowUpdatingEventArgs properties provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand to execute when performing the Update.
Errors	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType	Gets the type of SQL statement to execute.
Status	Gets or sets the UpdateStatus of the Command property.
TableMapping	Gets the DataTableMapping to send through the Update.

See Also

[MySqlConnection Class](#) , [MySqlConnection Namespace](#)

23.2.4.1.3.1.2.2.1.1.7.1. MySqlConnectionRowUpdatingEventHandler Delegate

Represents the method that will handle the RowUpdating event of a

[MySqlDataAdapter](#) .

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatingEventHandler( _  
    ByVal sender As Object, _  
    ByVal e As MySqlRowUpdatingEventArgs _  
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatingEventHandler(  
objectsender,  
MySqlRowUpdatingEventArgs  
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.7.1.1. [MySqlRowUpdatingEventArgs Class](#)

Provides data for the RowUpdating event. This class cannot be inherited.

For a list of all members of this type, see [MySqlRowUpdatingEventArgs Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatingEventArgs_  
    Inherits RowUpdatingEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatingEventArgs : RowUpdatingEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlRowUpdatingEventArgs Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.7.1.1.1.1. MySqlRowUpdatingEventArgs Members

[MySqlRowUpdatingEventArgs overview](#)

Public Instance Constructors

MySqlRowUpdatingEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatingEventArgs class.
---	--

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand to execute when performing the Update.
Errors(inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row(inherited from RowUpdatingEventArgs)	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType(inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status(inherited from RowUpdatingEventArgs)	Gets or sets the UpdateStatus of the Command property.
TableMapping(inherited from)	Gets the DataTableMapping to send through the

RowUpdatingEventArgs) Update.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
ToString(inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLRowUpdatingEventArgs Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.2.2.1.1.7.1.1.1.1. MySQLRowUpdatingEventArgs Constructor

Initializes a new instance of the MySQLRowUpdatingEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _  
    ByVal row As DataRow, _  
    ByVal command As IDbCommand, _  
    ByVal statementType As StatementType, _  
    ByVal tableMapping As DataTableMapping _  
)
```

Syntax: C#

```
public MySQLRowUpdatingEventArgs(  
    DataRow row,  
    IDbCommand command,  
    StatementType statementType,  
    DataTableMapping tableMapping  
) ;
```

Parameters

- row: The DataRow to Update.
- command: The IDbCommand to execute during Update.
- statementType: One of the StatementType values that specifies the type of query executed.
- tableMapping: The DataTableMapping sent through an Update.

See Also

[MySQLRowUpdatingEventArgs Class](#) , [MySQL.Data.MySqlCommand Namespace](#)

23.2.4.1.3.1.2.2.1.1.7.1.1.1.2. Command Property

Gets or sets the MySqlCommand to execute when performing the Update.

Syntax: Visual Basic

Overloads Public Property Command As MySqlCommand

Syntax: C#

```
new public MySqlCommand Command {get; set;}
```

See Also

[MySQLRowUpdatingEventArgs Class](#) , [MySQL.Data.MySqlCommand Namespace](#)

23.2.4.1.3.1.2.3. MySqlCommandBuilder Constructor (MySqlDataAdapter, Boolean)

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal adapter As MySqlDataAdapter, _
    ByVal lastOneWins As Boolean _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
    MySqlDataAdapter adapter,
```

```
boollastOneWins  
);
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLCommandBuilder Constructor Overload List](#)

23.2.4.1.3.1.2.4. MySQLCommandBuilder Constructor (Boolean)

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal lastOneWins As Boolean _  
)
```

Syntax: C#

```
public MySQLCommandBuilder(  
boollastOneWins  
);
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLCommandBuilder Constructor Overload List](#)

23.2.4.1.3.1.3. DataAdapter Property

Syntax: Visual Basic

```
Public Property DataAdapter As MySQLDataAdapter
```

Syntax: C#

```
public MySQLDataAdapter DataAdapter {get; set;}
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.4. QuotePrefix Property

Syntax: Visual Basic

```
Public Property QuotePrefix As String
```

Syntax: C#

```
public string QuotePrefix {get; set;}
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.5. QuoteSuffix Property

Syntax: Visual Basic

```
Public Property QuoteSuffix As String
```

Syntax: C#

```
public string QuoteSuffix {get; set;}
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.6. MySqlCommandBuilder.GetDeleteCommand Method

Syntax: Visual Basic

```
Public Function GetDeleteCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetDeleteCommand();
```

See Also

[MySQLCommandBuilder Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.7. MySqlCommandBuilder.GetInsertCommand Method

Syntax: Visual Basic

```
Public Function GetInsertCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetInsertCommand();
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.8. MySqlCommandBuilder.GetUpdateCommand Method

Syntax: Visual Basic

```
Public Function GetUpdateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetUpdateCommand();
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.3.1.9. MySqlCommandBuilder.RefreshSchema Method

Syntax: Visual Basic

```
Public Sub RefreshSchema()
```

Syntax: C#

```
public void RefreshSchema();
```

See Also

[MySqlCommandBuilder Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.4. MySqlException Class

The exception that is thrown when MySQL returns an error. This class cannot be inherited.

For a list of all members of this type, see [MySqlException Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlException_  
    Inherits SystemException
```

Syntax: C#

```
public sealed class MySqlException : SystemException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlException Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.4.1. **MySqlException Members**

[MySqlException overview](#)

Public Instance Properties

Data(inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink(inherited from Exception)	Gets or sets a link to the help file associated with this exception.

InnerException(inherited from Exception)	Gets the Exception instance that caused the current exception.
Message(inherited from Exception)	Gets a message that describes the current exception.
Number	Gets a number that identifies the type of error.
Source(inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace(inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite(inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException(inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData(inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType(inherited from Exception)	Gets the runtime type of the current instance.
ToString(inherited from Exception)	Creates and returns a string representation of the current exception.

See Also

[MySQLException Class](#) , [MySQL.Data.MySqlClient Namespace](#)

Gets a number that identifies the type of error.

Syntax: Visual Basic

```
Public ReadOnly Property Number As Integer
```

Syntax: C#

```
public int Number {get;}
```

See Also

[MySQLException Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.5. MySqlHelper Class

Helper class that makes it easier to work with the provider.

For a list of all members of this type, see [MySqlHelper Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlHelper
```

Syntax: C#

```
public sealed class MySqlHelper
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySqlHelper Members](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.5.1. MySqlHelper Members

[MySqlHelper overview](#)

Public Static (Shared) Methods

ExecuteDataRow	Executes a single SQL command and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteDataset	Overloaded. Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteNonQuery	Overloaded. Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.
ExecuteReader	Overloaded. Executes a single command against a MySQL database.
ExecuteScalar	Overloaded. Execute a single command against a MySQL database.
UpdateDataSet	Updates the given table with data from the given DataSet

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
ToString(inherited from Object)	Returns a String that represents the current Object.

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.5.1.1. MySQLHelper.ExecuteDataRow Method

Executes a single SQL command and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Public Shared Function ExecuteDataRow( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray parms As MySqlParameter() _  
) As DataRow
```

Syntax: C#

```
public static DataRow ExecuteDataRow(  
    string connectionString,  
    string commandText,  
    params MySqlParameter[] parms  
);
```

Parameters

- connectionString: Settings to be used for the connection
- commandText: Command to execute
- parms: Parameters to use for the command

Return Value

DataRow containing the first row of the resultset

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.5.1.2. ExecuteDataset Method

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySQLConnection](#) object remains unchanged after execution of this method.

Overload List

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySQLConnection](#) object remains unchanged after execution of this method.

- [public static DataSet ExecuteDataset\(MySqlConnection,string\);](#)

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySQLConnection](#) object remains unchanged after execution of this method.

- [public static DataSet ExecuteDataset\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- [public static DataSet ExecuteDataset\(string,string\);](#)

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- [public static DataSet ExecuteDataset\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.5.1.2.1. MySQLHelper.ExecuteDataset Method (MySqlConnection, String)

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySQLConnection](#) object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    MySqlConnection connection,  
    string commandText  
);
```

Parameters

- connection: [MySqlConnection](#) object to use
- commandText: Command to execute

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLHelper.ExecuteDataset Overload List](#)

23.2.4.1.5.1.2.2. MySQLHelper.ExecuteDataset Method (MySqlConnection, String, MySqlParameter[])

Executes a single SQL command and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    MySqlConnection connection,  
    string commandText,  
    params MySqlParameter[] commandParameters  
);
```

Parameters

- connection: [MySqlConnection](#) object to use
- commandText: Command to execute
- commandParameters: Parameters to use for the command

Return Value

DataSet containing the resultset

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlHelper.ExecuteDataset Overload List](#)

23.2.4.1.5.1.2.3. [MySqlHelper.ExecuteDataset Method \(String, String\)](#)

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connectionString As String, _  
    ByVal commandText As String _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
    string connectionString,  
    string commandText  
);
```

Parameters

- connectionString: Settings to be used for the connection
- commandText: Command to execute

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLHelper.ExecuteDataset Overload List](#)

23.2.4.1.5.1.2.4. MySQLHelper.ExecuteDataset Method (String, String, MySqlParameter[])

Executes a single SQL command and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(  
stringconnectionString,  
stringcommandText,  
    params MySqlParameter[]commandParameters  
);
```

Parameters

- connectionString: Settings to be used for the connection
- commandText: Command to execute
- commandParameters: Parameters to use for the command

Return Value

DataSet containing the resultset

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlHelper.ExecuteDataset Overload List](#)

23.2.4.1.5.1.3. ExecuteNonQuery Method

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

Overload List

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

- [public static int ExecuteNonQuery\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Executes a single command against a MySQL database. A new [MySqlConnection](#) is created using the [ConnectionString](#) given.

- [public static int ExecuteNonQuery\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.5.1.3.1. MySqlHelper.ExecuteNonQuery Method (MySqlConnection, String, MySqlParameter[])

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(  
MySqlConnection connection,  
string commandText,  
    params MySqlParameter[] commandParameters  
);
```

Parameters

- connection: [MySqlConnection](#) object to use
- commandText: SQL command to be executed
- commandParameters: Array of [MySqlParameter](#) objects to use with the command.

Return Value

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlHelper.ExecuteNonQuery Overload List](#)

23.2.4.1.5.1.3.2. [MySqlHelper.ExecuteNonQuery Method \(String, String, MySqlParameter\[\]\)](#)

Executes a single command against a MySQL database. A new [MySqlConnection](#) is created using the [ConnectionString](#) given.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray parms As MySqlParameter() _  
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(  
    string connectionString,  
    string commandText,  
    params MySqlParameter[] parms  
);
```

Parameters

- connectionString: [ConnectionString](#) to use
- commandText: SQL command to be executed
- parms: Array of [MySqlParameter](#) objects to use with the command.

Return Value

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlHelper.ExecuteNonQuery Overload List](#)

23.2.4.1.5.1.4. ExecuteReader Method

Executes a single command against a MySQL database.

Overload List

Executes a single command against a MySQL database.

- [public static MySqlDataReader ExecuteReader\(string,string\);](#)

Executes a single command against a MySQL database.

- [public static MySqlDataReader ExecuteReader\(string,string,params
MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.5.1.4.1. MySqlHelper.ExecuteReader Method (String, String)

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _  
    ByVal connectionString As String, _  
    ByVal commandText As String _  
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(  
stringconnectionString,  
stringcommandText  
);
```

Parameters

- connectionString: Settings to use for this command
- commandText: Command text to use

Return Value

[MySqlDataReader](#) object ready to read the results of the command

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlHelper.ExecuteReader Overload List](#)

23.2.4.1.5.1.4.2. MySqlHelper.ExecuteReader Method (String, String, MySqlParameter[])

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _
```

) As MySqlDataReader

Syntax: C#

```
public static MySqlDataReader ExecuteReader(  
    string connectionString,  
    string commandText,  
    params MySqlParameter[] commandParameters  
);
```

Parameters

- connectionString: Settings to use for this command
- commandText: Command text to use
- commandParameters: Array of [MySqlParameter](#) objects to use with the command

Return Value

[MySqlDataReader](#) object ready to read the results of the command

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlHelper.ExecuteReader Overload List](#)

23.2.4.1.5.1.5. ExecuteScalar Method

Execute a single command against a MySQL database.

Overload List

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(MySqlConnection,string\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(string,string\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#)

23.2.4.1.5.1.5.1. [MySqlHelper.ExecuteScalar Method \(MySqlConnection, String\)](#)

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
    MySqlConnection connection,  
    string commandText  
);
```

Parameters

- connection: [MySqlConnection](#) object to use
- commandText: Command text to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLHelper.ExecuteScalar Overload List](#)

23.2.4.1.5.1.5.2. MySQLHelper.ExecuteScalar Method (MySQLConnection, String, MySQLParameter[])

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connection As MySqlConnection, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySQLParameter() _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
MySQLConnection connection,  
string commandText,  
    params MySQLParameter[] commandParameters  
);
```

Parameters

- connection: [MySQLConnection](#) object to use
- commandText: Command text to use for the command
- commandParameters: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLHelper.ExecuteScalar Overload List](#)

23.2.4.1.5.1.5.3. MySQLHelper.ExecuteScalar Method (String, String)

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connectionString As String, _  
    ByVal commandText As String _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
stringconnectionString,  
stringcommandText  
);
```

Parameters

- connectionString: Settings to use for the update
- commandText: Command text to use for the update

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#) ,
[MySQLHelper.ExecuteScalar Overload List](#)

23.2.4.1.5.1.5.4. MySQLHelper.ExecuteScalar Method (String, String, MySqlParameter[])

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ParamArray commandParameters As MySqlParameter() _  
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(  
    string connectionString,  
    string commandText,  
    params MySqlParameter[] commandParameters  
);
```

Parameters

- connectionString: Settings to use for the command
- commandText: Command text to use for the command
- commandParameters: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#) , [MySql.Data.MySqlClient Namespace](#) ,
[MySqlHelper.ExecuteScalar Overload List](#)

23.2.4.1.5.1.6. MySqlHelper.UpdateDataSet Method

Updates the given table with data from the given DataSet

Syntax: Visual Basic

```
Public Shared Sub UpdateDataSet( _  
    ByVal connectionString As String, _  
    ByVal commandText As String, _  
    ByVal ds As DataSet, _  
    ByVal tablename As String _  
)
```

Syntax: C#

```
public static void UpdateDataSet(  
    string connectionString,
```

```
stringcommandText,  
DataSets,  
stringtablename  
);
```

Parameters

- connectionString: Settings to use for the update
- commandText: Command text to use for the update
- ds: DataSetcontaining the new data to use in the update
- tablename: Tablename in the dataset to update

See Also

[MySQLHelper Class](#) , [MySQL.Data.MySqlClient Namespace](#)

23.2.4.1.6. MySQLErrorCode Enumeration

Syntax: Visual Basic

```
Public Enum MySQLErrorCode
```

Syntax: C#

```
public enum MySQLErrorCode
```

Members

Member Name	Description
PacketTooLarge	
PasswordNotAllowed	
DuplicateKeyEntry	
HostNotPrivileged	
PasswordNoMatch	
AnonymousUser	
DuplicateKey	

KeyNotFound	
DuplicateKeyName	

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

23.2.4.2. MySql.Data.Types

[Namespace hierarchy](#)

Classes

Class	Description
MySqlConnectionException	Summary description for MySqlConnectionException.
MySqlDateTime	Summary description for MySqlDateTime.
MySqlValue	

23.2.4.2.1. MySql.Data.TypesHierarchy

See Also

[MySql.Data.Types Namespace](#)

23.2.4.2.2. MySqlConnectionException Class

Summary description for MySqlConnectionException.

For a list of all members of this type, see [MySqlConnectionException Members](#) .

Syntax: Visual Basic

Public Class MySqlConnectionException_
Inherits ApplicationException

Syntax: C#

```
public class MySqlConnectionException : ApplicationException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.Types](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlConnectionException Members](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.2.1. MySqlConnectionException Members

[MySqlConnectionException overview](#)

Public Instance Constructors

MySqlConnectionException Constructor	Ctor
--	------

Public Instance Properties

Data(inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink(inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException(inherited from Exception)	Gets the Exception instance that caused the current exception.
Message(inherited from	

Exception)	Gets a message that describes the current exception.
Source(inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace(inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite(inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException(inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData(inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType(inherited from Exception)	Gets the runtime type of the current instance.
ToString(inherited from Exception)	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult(inherited from Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.
-----------------------------------	---

Protected Instance Methods

Finalize(inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
---------------------------------	--

<code>MemberwiseClone(inherited from Object)</code>	Creates a shallow copy of the current Object.
---	---

See Also

[MySQLConversionException Class](#) , [MySQL.Data.Types Namespace](#)

23.2.4.2.2.1.1. MySQLConversionException Constructor

Syntax: Visual Basic

```
Public Sub New( _  
    ByVal msg As String _  
)
```

Syntax: C#

```
public MySQLConversionException(  
stringmsg  
);
```

See Also

[MySQLConversionException Class](#) , [MySQL.Data.Types Namespace](#)

23.2.4.2.3. MySQLDateTime Class

Summary description for MySQLDateTime.

For a list of all members of this type, see [MySQLDateTime Members](#) .

Syntax: Visual Basic

```
Public Class MySQLDateTime_  
    Inherits MySQLValue_  
    Implements IConvertible, IComparable
```

Syntax: C#

```
public class MySQLDateTime : MySQLValue, IConvertible, IComparable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.Types](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDateTime Members](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1. MySqlDateTime Members

[MySqlDateTime overview](#)

Public Static (Shared) Type Conversions

[Explicit MySqlDateTime to DateTime Conversion](#)

Public Instance Properties

Day	Returns the day portion of this datetime
Hour	Returns the hour portion of this datetime
IsNull (inherited from MySqlValue)	
IsValidDateTime	Indicates if this object contains a value that can be represented as a DateTime
Minute	Returns the minute portion of this datetime
Month	Returns the month portion of this datetime
Second	Returns the second portion of this datetime
ValueAsObject (inherited from MySqlValue)	Returns the value of this field as an object
Year	Returns the year portion of this datetime

Public Instance Methods

Equals(inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetDateTime	Returns this value as a DateTime
GetHashCode(inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType(inherited from Object)	Gets the Type of the current instance.
ToString	Returns a MySQL specific string representation of this value

Protected Instance Fields

classType (inherited from MySqlValue)	The system type represented by this value
dbType (inherited from MySqlValue)	The generic dbtype of this value
isNull (inherited from MySqlValue)	Is this value null
mysqlDbType (inherited from MySqlValue)	The specific MySQL db type
mysqlTypeName (inherited from MySqlValue)	The MySQL specific typename of this value
objectValue (inherited from MySqlValue)	

Protected Instance Methods

Finalize(inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.1. MySqlDateTime Explicit MySqlDateTime to DateTime Conversion

Syntax: Visual Basic

```
MySqlDateTime.op_Explicit(val)
```

Syntax: C#

```
public static explicit operator DateTime(  
MySqlDateTime val  
);
```

Parameters

- val:

Return Value

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.2. Day Property

Returns the day portion of this datetime

Syntax: Visual Basic

```
Public Property Day As Integer
```

Syntax: C#

```
public int Day {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.3. Hour Property

Returns the hour portion of this datetime

Syntax: Visual Basic

```
Public Property Hour As Integer
```

Syntax: C#

```
public int Hour {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4. IsNull Property

Syntax: Visual Basic

```
Public Property IsNull As Boolean
```

Syntax: C#

```
public bool IsNull {get; set;}
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4.1. MySqlValue Class

For a list of all members of this type, see [MySqlValue Members](#) .

Syntax: Visual Basic

```
MustInherit Public Class MySqlValue
```

Syntax: C#

```
public abstract class MySqlValue
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.Types](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlValue Members](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1. MySqlValue Members

[MySqlValue overview](#)

Protected Static (Shared) Fields

numberFormat

Public Instance Constructors

MySqlValue Constructor	Initializes a new instance of the MySqlValue class.
--	---

Public Instance Properties

IsNull	
ValueAsObject	Returns the value of this field as an object

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.

GetType(inherited from Object)	Gets the Type of the current instance.
ToString	Returns a string representation of this value

Protected Instance Fields

classType	The system type represented by this value
dbType	The generic dbtype of this value
isNull	Is this value null
mySqlDbType	The specific MySQL db type
mySqlTypeName	The MySQL specific typename of this value
objectValue	

Protected Instance Methods

Finalize(inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone(inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.1. [MySqlValue.numberFormat Field](#)

Syntax: Visual Basic

Protected Shared numberFormat As NumberFormatInfo

Syntax: C#

```
protected static NumberFormatInfo numberFormat;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.2. MySqlValue Constructor

Initializes a new instance of the [MySqlValue](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlValue();
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.3. ValueAsObject Property

Returns the value of this field as an object

Syntax: Visual Basic

```
Public ReadOnly Property ValueAsObject As Object
```

Syntax: C#

```
public object ValueAsObject {get;}
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.4. MySqlValue.ToString Method

Returns a string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySQLValue Class](#) , [MySQL.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.5. MySQLValue.classType Field

The system type represented by this value

Syntax: Visual Basic

```
Protected classType As Type
```

Syntax: C#

```
protected Type classType;
```

See Also

[MySQLValue Class](#) , [MySQL.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.6. MySQLValue.dbType Field

The generic dbtype of this value

Syntax: Visual Basic

```
Protected dbType As DbType
```

Syntax: C#

```
protected DbType dbType;
```

See Also

[MySQLValue Class](#) , [MySQL.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.7. MySQLValue.mysqlDbType Field

The specific MySQL db type

Syntax: Visual Basic

```
Protected mySqlDbType As MySqlDbType
```

Syntax: C#

```
protected MySqlDbType mySqlDbType;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.8. [MySqlValue.mySqlTypeName](#) Field

The MySQL specific typename of this value

Syntax: Visual Basic

```
Protected mySqlTypeName As String
```

Syntax: C#

```
protected string mySqlTypeName;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.4.1.1.9. [MySqlValue.objectValue](#) Field

Syntax: Visual Basic

```
Protected objectValue As Object
```

Syntax: C#

```
protected object objectValue;
```

See Also

[MySqlValue Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.5. IsValidDateTime Property

Indicates if this object contains a value that can be represented as a DateTime

Syntax: Visual Basic

```
Public ReadOnly Property IsValidDateTime As Boolean
```

Syntax: C#

```
public bool IsValidDateTime {get;}
```

See Also

[MySQLDateTime Class](#) , [MySQL.Data.Types Namespace](#)

23.2.4.2.3.1.6. Minute Property

Returns the minute portion of this datetime

Syntax: Visual Basic

```
Public Property Minute As Integer
```

Syntax: C#

```
public int Minute {get; set;}
```

See Also

[MySQLDateTime Class](#) , [MySQL.Data.Types Namespace](#)

23.2.4.2.3.1.7. Month Property

Returns the month portion of this datetime

Syntax: Visual Basic

```
Public Property Month As Integer
```

Syntax: C#

```
public int Month {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.8. Second Property

Returns the second portion of this datetime

Syntax: Visual Basic

```
Public Property Second As Integer
```

Syntax: C#

```
public int Second {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.9. Year Property

Returns the year portion of this datetime

Syntax: Visual Basic

```
Public Property Year As Integer
```

Syntax: C#

```
public int Year {get; set;}
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.10. MySqlDateTime.GetDateTime Method

Returns this value as a DateTime

Syntax: Visual Basic

```
Public Function GetDateTime() As Date
```

Syntax: C#

```
public DateTime GetDateTime();
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.4.2.3.1.11. MySqlDateTime.ToString Method

Returns a MySQL specific string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySqlDateTime Class](#) , [MySql.Data.Types Namespace](#)

23.2.5. Connector/NET Notes and Tips

In this section we will cover some of the more common use cases for Connector/NET, including BLOB handling, date handling, and using Connector/NET with common tools such as Crystal Reports.

23.2.5.1. Connecting to MySQL Using Connector/NET

23.2.5.1.1. Introduction

All interaction between a .NET application and the MySQL server is routed through a `MySqlConnection` object. Before your application can interact with the server, a `MySqlConnection` object must be instanced, configured, and opened.

Even when using the `MySQLHelper` class, a `MySQLConnection` object is created by the helper class.

In this section, we will describe how to connect to MySQL using the `MySQLConnection` object.

23.2.5.1.2. Creating a Connection String

The `MySQLConnection` object is configured using a connection string. A connection string contains several key/value pairs, separated by semicolons. Each key/value pair is joined with an equals sign.

The following is a sample connection string:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

In this example, the `MySQLConnection` object is configured to connect to a MySQL server at `127.0.0.1`, with a username of `root` and a password of `12345`. The default database for all statements will be the `test` database.

The following options are typically used (a full list of options is available in the API documentation for [Section 23.2.3.3.15, “ConnectionString”](#)):

- **Server:** The name or network address of the instance of MySQL to which to connect. The default is `localhost`. Aliases include `host`, `Data Source`, `DataSource`, `Address`, `Addr` and `Network Address`.
- **uid:** The MySQL user account to use when connecting. Aliases include `User Id`, `Username` and `User name`.
- **Pwd:** The password for the MySQL account being used. Alias `Password` can also be used.
- **Database:** The default database that all statements are applied to. Default is `mysql`. Alias `Initial Catalog` can also be used.
- **Port:** The port MySQL is using to listen for connections. Default is `3306`. Specify `-1` for this value to use a named-pipe connection.

23.2.5.1.3. Opening a Connection

Once you have created a connection string it can be used to open a connection to the MySQL server.

The following code is used to create a MySqlConnection object, assign the connection string, and open the connection.

Visual Basic Example

```
Dim conn As New MySql.Data.MySqlClient.MySqlConnection
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"

Try
    conn.ConnectionString = myConnectionString
    conn.Open()

Catch ex As MySql.Data.MySqlClient.MySqlException
    MessageBox.Show(ex.Message)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection();
    conn.ConnectionString = myConnectionString;
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

You can also pass the connection string to the constructor of the MySqlConnection class:

Visual Basic Example

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnect
        conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    MessageBox.Show(ex.Message)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionSt
        conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

Once the connection is open it can be used by the other Connector/.NET classes to communicate with the MySQL server.

23.2.5.1.4. Handling Connection Errors

Because connecting to an external server is unpredictable, it is important to add error handling to your .NET application. When there is an error connecting, the MySqlConnection class will return a MySqlException object. This object has two properties that are of interest when handling errors:

- Message: A message that describes the current exception.

- Number: The MySQL error number.

When handling errors, you can your application's response based on the error number. The two most common error numbers when connecting are as follows:

- 0: Cannot connect to server.
- 1045: Invalid username and/or password.

The following code shows how to adapt the application's response based on the actual error:

Visual Basic Example

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnect
conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    Select Case ex.Number
        Case 0
            MessageBox.Show("Cannot connect to server. Contact admin
        Case 1045
            MessageBox.Show("Invalid username/password, please try a
    End Select
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionSt
conn.Open();
```

```

}
catch (MySQL.Data.MySqlClient.MySqlException ex)
{
switch (ex.Number)
{
case 0:
    MessageBox.Show("Cannot connect to server. Contact admin");
case 1045:
    MessageBox.Show("Invalid username/password, please try again");
}
}
}

```

Important: Note that if you are using multilanguage databases you must specify the character set in the connection string. If you do not specify the character set, the connection defaults to the latin1 charset. You can specify the character set as part of the connection string, for example:

```

MySQLConnection myConnection = new MySQLConnection("server=127.0.0.1
    "pwd=12345;database=test;Charset=latin1;");

```

23.2.5.2. Using the Connector/NET with Prepared Statements

23.2.5.2.1. Introduction

As of MySQL 4.1, it is possible to use prepared statements with Connector/NET. Use of prepared statements can provide significant performance improvements on queries that are executed more than once.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

23.2.5.2.2. Preparing Statements in Connector/NET

To prepare a statement, create a command object and set the `.CommandText` property to your query.

After entering your statement, call the `.Prepare` method of the `MySQLCommand` object. After the statement is prepared, add parameters for each of the dynamic elements in the query.

After you enter your query and enter parameters, execute the statement using the `.ExecuteNonQuery()`, `.ExecuteScalar()`, or `.ExecuteReader` methods.

For subsequent executions, you need only modify the values of the parameters and call the execute method again, there is no need to set the `.CommandText` property or redefine the parameters.

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = strConnection

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, ?number, ?text)"
    cmd.Prepare()

    cmd.Parameters.Add("?number", 1)
    cmd.Parameters.Add("?text", "One")

    For i = 1 To 1000
        cmd.Parameters["?number"].Value = i
        cmd.Parameters["?text"].Value = "A string value"

        cmd.ExecuteNonQuery()
    Next
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message)
End Try
```

C# Example

```
MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();
```

```

conn.ConnectionString = strConnection;

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, ?number, ?te
    cmd.Prepare();

    cmd.Parameters.Add("?number", 1);
    cmd.Parameters.Add("?text", "One");

    for (int i=1; i <= 1000; i++)
    {
        cmd.Parameters["?number"].Value = i;
        cmd.Parameters["?text"].Value = "A string value";

        cmd.ExecuteNonQuery();
    }
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Me
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

23.2.5.3. Accessing Stored Procedures with Connector/NET

23.2.5.3.1. Introduction

With the release of MySQL version 5 the MySQL server now supports stored procedures with the SQL 2003 stored procedure syntax.

A stored procedure is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored procedure instead.

Stored procedures can be particularly useful in situations such as the following:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures for

all common operations. This provides a consistent and secure environment, and procedures can ensure that each operation is properly logged. In such a setup, applications and users would not get any access to the database tables directly, but can only execute specific stored procedures.

Connector/NET supports the calling of stored procedures through the `MySQLCommand` object. Data can be passed in and out of a MySQL stored procedure through use of the `MySQLCommand.Parameters` collection.

Note: When you call a stored procedure, the command object makes an additional `SELECT` call to determine the parameters of the stored procedure. You must ensure that the user calling the procedure has the `SELECT` privilege on the `mysql.proc` table to enable them to verify the parameters. Failure to do this will result in an error when calling the procedure.

This section will not provide in-depth information on creating Stored Procedures. For such information, please refer to <http://dev.mysql.com/doc/mysql/en/stored-procedures.html>.

A sample application demonstrating how to use stored procedures with Connector/NET can be found in the `Samples` directory of your Connector/NET installation.

23.2.5.3.2. Creating Stored Procedures from Connector/NET

Stored procedures in MySQL can be created using a variety of tools. First, stored procedures can be created using the **mysql** command-line client. Second, stored procedures can be created using the MySQL Query Browser GUI client. Finally, stored procedures can be created using the `.ExecuteNonQuery` method of the `MySQLCommand` object:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"
```

```

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "CREATE PROCEDURE add_emp(" _
        & "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATET
        & "BEGIN INSERT INTO emp(first_name, last_name, birthdate) "
        & "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT

    cmd.ExecuteNonQuery()
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Me
End Try

```

C# Example

```

MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "CREATE PROCEDURE add_emp(" +
        "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIM
        "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " +
        "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_I

    cmd.ExecuteNonQuery();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Messag
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

It should be noted that, unlike the command-line and GUI clients, you are not required to specify a special delimiter when creating stored procedures in Connector/NET.

23.2.5.3.3. Calling a Stored Procedure from Connector/NET

To call a stored procedure using Connector/NET, create a MySqlCommand object and pass the stored procedure name as the .CommandText property. Set the .CommandType property to CommandType.StoredProcedure.

After the stored procedure is named, create one MySqlCommand parameter for every parameter in the stored procedure. IN parameters are defined with the parameter name and the object containing the value, OUT parameters are defined with the parameter name and the datatype that is expected to be returned. All parameters need the parameter direction defined.

After defining parameters, call the stored procedure by using the MySqlCommand.ExecuteNonQuery() method:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "add_emp"
    cmd.CommandType = CommandType.StoredProcedure

    cmd.Parameters.Add("?lname", 'Jones')
    cmd.Parameters["?lname"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("?fname", 'Tom')
    cmd.Parameters["?fname"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("?bday", #12/13/1977 2:17:36 PM#)
    cmd.Parameters["?bday"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("?empno", MySqlDbType.Int32)
    cmd.Parameters["?empno"].Direction = ParameterDirection.Output

    cmd.ExecuteNonQuery()

    MessageBox.Show(cmd.Parameters["?empno"].Value)
Catch ex As MySqlException
```

```
        MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Me  
End Try
```

C# Example

```
MySQL.Data.MySqlClient.MySqlConnection conn;  
MySQL.Data.MySqlClient.MySqlCommand cmd;  
  
conn = new MySQL.Data.MySqlClient.MySqlConnection();  
cmd = new MySQL.Data.MySqlClient.MySqlCommand();  
  
conn.ConnectionString = "server=127.0.0.1;uid=root;" +  
    "pwd=12345;database=test;";  
  
try  
{  
    conn.Open();  
    cmd.Connection = conn;  
  
    cmd.CommandText = "add_emp";  
    cmd.CommandType = CommandType.StoredProcedure;  
  
    cmd.Parameters.Add("?lname", "Jones");  
    cmd.Parameters["?lname"].Direction = ParameterDirection.Input;  
  
    cmd.Parameters.Add("?fname", "Tom");  
    cmd.Parameters["?fname"].Direction = ParameterDirection.Input;  
  
    cmd.Parameters.Add("?bday", DateTime.Parse("12/13/1977 2:17:36 P  
    cmd.Parameters["?bday"].Direction = ParameterDirection.Input;  
  
    cmd.Parameters.Add("?empno", MySqlDbType.Int32);  
    cmd.Parameters["?empno"].Direction = ParameterDirection.Output;  
  
    cmd.ExecuteNonQuery();  
  
    MessageBox.Show(cmd.Parameters["?empno"].Value);  
}  
catch (MySQL.Data.MySqlClient.MySqlException ex)  
{  
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Me  
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);  
}
```

Once the stored procedure is called, the values of output parameters can be retrieved by using the `.Value` property of the `MySQLConnector.Parameters` collection.

23.2.5.4. Handling BLOB Data With Connector/NET

23.2.5.4.1. Introduction

One common use for MySQL is the storage of binary data in BLOB columns. MySQL supports four different BLOB datatypes: TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB.

Data stored in a BLOB column can be accessed using Connector/NET and manipulated using client-side code. There are no special requirements for using Connector/NET with BLOB data.

Simple code examples will be presented within this section, and a full sample application can be found in the `Samples` directory of the Connector/NET installation.

23.2.5.4.2. Preparing the MySQL Server

The first step in using MySQL with BLOB data is to configure the server. Let's start by creating a table to be accessed. In my file tables, I usually have four columns: an `AUTO_INCREMENT` column of appropriate size (`UNSIGNED SMALLINT`) to serve as a primary key to identify the file, a `VARCHAR` column that stores the filename, an `UNSIGNED MEDIUMINT` column that stores the size of the file, and a `MEDIUMBLOB` column that stores the file itself. For this example, I will use the following table definition:

```
CREATE TABLE file(  
file_id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,  
file_name VARCHAR(64) NOT NULL,  
file_size MEDIUMINT UNSIGNED NOT NULL,  
file MEDIUMBLOB NOT NULL);
```

After creating a table, you may need to modify the `max_allowed_packet` system variable. This variable determines how large of a packet (i.e. a single row) can be sent to the MySQL server. By default, the server will only accept a maximum size of 1 meg from our client application. If you do not intend to exceed 1 meg, this should be fine. If you do intend to exceed 1 meg in your file transfers, this number has to be increased.

The `max_allowed_packet` option can be modified using MySQL Administrator's

Startup Variables screen. Adjust the Maximum allowed option in the Memory section of the Networking tab to an appropriate setting. After adjusting the value, click the Apply Changes button and restart the server using the Service Control screen of MySQL Administrator. You can also adjust this value directly in the my.cnf file (add a line that reads `max_allowed_packet=xxM`), or use the `SET max_allowed_packet=xxM;` syntax from within MySQL.

Try to be conservative when setting `max_allowed_packet`, as transfers of BLOB data can take some time to complete. Try to set a value that will be adequate for your intended use and increase the value if necessary.

23.2.5.4.3. Writing a File to the Database

To write a file to a database we need to convert the file to a byte array, then use the byte array as a parameter to an `INSERT` query.

The following code opens a file using a `FileStream` object, reads it into a byte array, and inserts it into the `file` table:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

Dim SQL As String

Dim fileSize As UInt32
Dim rawData() As Byte
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    fs = New FileStream("c:\image.png", FileMode.Open, FileAccess.Read)
    fileSize = fs.Length

    rawData = New Byte(fileSize) {}
    fs.Read(rawData, 0, fileSize)
    fs.Close()

    conn.Open()
```

```

SQL = "INSERT INTO file VALUES(NULL, ?FileName, ?FileSize, ?File

cmd.Connection = conn
cmd.CommandText = SQL
cmd.Parameters.Add("?FileName", strFileName)
cmd.Parameters.Add("?FileSize", FileSize)
cmd.Parameters.Add("?File", rawData)

cmd.ExecuteNonQuery()

MessageBox.Show("File Inserted into database successfully!", _
"Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)

conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    fs = new FileStream(@"c:\image.png", FileMode.Open, FileAccess.R
    FileSize = fs.Length;

    rawData = new byte[FileSize];
    fs.Read(rawData, 0, FileSize);
    fs.Close();

    conn.Open();

    SQL = "INSERT INTO file VALUES(NULL, ?FileName, ?FileSize, ?File

```

```

cmd.Connection = conn;
cmd.CommandText = SQL;
cmd.Parameters.Add("?FileName", strFileName);
cmd.Parameters.Add("?FileSize", FileSize);
cmd.Parameters.Add("?File", rawData);

cmd.ExecuteNonQuery();

MessageBox.Show("File Inserted into database successfully!",
    "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Me
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

The Read method of the FileStream object is used to load the file into a byte array which is sized according to the Length property of the FileStream object.

After assigning the byte array as a parameter of the MySqlCommand object, the ExecuteNonQuery method is called and the BLOB is inserted into the file table.

23.2.5.4.4. Reading a BLOB from the Database to a File on Disk

Once a file is loaded into the file table, we can use the MySqlDataReader class to retrieve it.

The following code retrieves a row from the file table, then loads the data into a FileStream object to be written to disk:

Visual Basic Example

```

Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myData As MySqlDataReader
Dim SQL As String
Dim rawData() As Byte
Dim FileSize As UInt32
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _

```

```
& "uid=root;" _  
& "pwd=12345;" _  
& "database=test"
```

```
SQL = "SELECT file_name, file_size, file FROM file"
```

```
Try
```

```
conn.Open()
```

```
cmd.Connection = conn
```

```
cmd.CommandText = SQL
```

```
myData = cmd.ExecuteReader
```

```
If Not myData.HasRows Then Throw New Exception("There are no BLOC
```

```
myData.Read()
```

```
FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"))
```

```
rawData = New Byte(FileSize) {}
```

```
myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSi
```

```
fs = New FileStream("C:\newfile.png", FileMode.OpenOrCreate, Fil
```

```
fs.Write(rawData, 0, FileSize)
```

```
fs.Close()
```

```
MessageBox.Show("File successfully written to disk!", "Success!"
```

```
myData.Close()
```

```
conn.Close()
```

```
Catch ex As Exception
```

```
MessageBox.Show("There was an error: " & ex.Message, "Error", Me
```

```
End Try
```

C# Example

```
MySQL.Data.MySqlClient.MySqlConnection conn;
```

```
MySQL.Data.MySqlClient.MySqlCommand cmd;
```

```
MySQL.Data.MySqlClient.MySqlDataReader myData;
```

```
conn = new MySQL.Data.MySqlClient.MySqlConnection();
```

```
cmd = new MySQL.Data.MySqlClient.MySqlCommand();
```

```
string SQL;
```

```
UInt32 FileSize;
```

```
byte[] rawData;
```

```
FileStream fs;
```

```

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

SQL = "SELECT file_name, file_size, file FROM file";

try
{
    conn.Open();

    cmd.Connection = conn;
    cmd.CommandText = SQL;

    myData = cmd.ExecuteReader();

    if (! myData.HasRows)
        throw new Exception("There are no BLOBs to save");

    myData.Read();

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"));
    rawData = new byte[FileSize];

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSi

    fs = new FileStream(@"C:\newfile.png", FileMode.OpenOrCreate, Fi
    fs.Write(rawData, 0, FileSize);
    fs.Close();

    MessageBox.Show("File successfully written to disk!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    myData.Close();
    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Me
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

After connecting, the contents of the file table are loaded into a `MySqlDataReader` object. The `GetBytes` method of the `MySqlDataReader` is used to load the BLOB into a byte array, which is then written to disk using a `FileStream` object.

The `GetOrdinal` method of the `MySqlDataReader` can be used to determine the

integer index of a named column. Use of the GetOrdinal method prevents errors if the column order of the SELECT query is changed.

23.2.5.5. Using Connector/NET with Crystal Reports

23.2.5.5.1. Introduction

Crystal Reports is a common tool used by Windows application developers to perform reporting and document generation. In this section we will show how to use Crystal Reports XI with MySQL and Connector/NET.

23.2.5.5.2. Creating a Data Source

When creating a report in Crystal Reports there are two options for accessing the MySQL data while designing your report.

The first option is to use Connector/ODBC as an ADO data source when designing your report. You will be able to browse your database and choose tables and fields using drag and drop to build your report. The disadvantage of this approach is that additional work must be performed within your application to produce a dataset that matches the one expected by your report.

The second option is to create a dataset in VB.NET and save it as XML. This XML file can then be used to design a report. This works quite well when displaying the report in your application, but is less versatile at design time because you must choose all relevant columns when creating the dataset. If you forget a column you must re-create the dataset before the column can be added to the report.

The following code can be used to create a dataset from a query and write it to disk:

Visual Basic Example

```
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
```

```
& "pwd=12345;" _  
& "database=world"
```

Try

```
conn.Open()  
cmd.CommandText = "SELECT city.name AS cityName, city.population  
    & "country.name, country.population, country.continent " _  
    & "FROM country, city ORDER BY country.continent, country.na  
cmd.Connection = conn
```

```
myAdapter.SelectCommand = cmd  
myAdapter.Fill(myData)
```

```
myData.WriteXml("C:\dataset.xml", XmlWriteMode.WriteSchema)
```

Catch ex As Exception

```
    MessageBox.Show(ex.Message, "Report could not be created", Messa
```

End Try

C# Example

```
DataSet myData = new DataSet();  
MySQL.Data.MySqlClient.MySqlConnection conn;  
MySQL.Data.MySqlClient.MySqlCommand cmd;  
MySQL.Data.MySqlClient.MySqlDataAdapter myAdapter;
```

```
conn = new MySQL.Data.MySqlClient.MySqlConnection();  
cmd = new MySQL.Data.MySqlClient.MySqlCommand();  
myAdapter = new MySQL.Data.MySqlClient.MySqlDataAdapter();
```

```
conn.ConnectionString = "server=127.0.0.1;uid=root;" +  
    "pwd=12345;database=test;";
```

try

```
{  
    cmd.CommandText = "SELECT city.name AS cityName, city.population A  
    "country.name, country.population, country.continent " +  
    "FROM country, city ORDER BY country.continent, country.name";  
    cmd.Connection = conn;
```

```
    myAdapter.SelectCommand = cmd;  
    myAdapter.Fill(myData);
```

```
    myData.WriteXml(@"C:\dataset.xml", XmlWriteMode.WriteSchema);
```

}

```
catch (MySQL.Data.MySqlClient.MySqlException ex)
```

```
{
```

```
    MessageBox.Show(ex.Message, "Report could not be created",  
    MessageBoxButtons.OK, MessageBoxIcon.Error);
```

}

The resulting XML file can be used as an ADO.NET XML datasource when designing your report.

If you choose to design your reports using Connector/ODBC, it can be downloaded from dev.mysql.com.

23.2.5.5.3. Creating the Report

For most purposes the Standard Report wizard should help with the initial creation of a report. To start the wizard, open Crystal Reports and choose the New > Standard Report option from the File menu.

The wizard will first prompt you for a data source. If you are using Connector/ODBC as your data source, use the OLEDB provider for ODBC option from the OLE DB (ADO) tree instead of the ODBC (RDO) tree when choosing a data source. If using a saved dataset, choose the ADO.NET (XML) option and browse to your saved dataset.

The remainder of the report creation process is done automatically by the wizard.

After the report is created, choose the Report Options... entry of the File menu. Un-check the Save Data With Report option. This prevents saved data from interfering with the loading of data within our application.

23.2.5.5.4. Displaying the Report

To display a report we first populate a dataset with the data needed for the report, then load the report and bind it to the dataset. Finally we pass the report to the crViewer control for display to the user.

The following references are needed in a project that displays a report:

- CrystalDecisions.CrystalReports.Engine
- CrystalDecisions.ReportSource
- CrystalDecisions.Shared

- CrystalDecisions.Windows.Forms

The following code assumes that you created your report using a dataset saved using the code shown in [Section 23.2.5.5.2, “Creating a Data Source”](#), and have a crViewer control on your form named myViewer.

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = _
    "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()

    cmd.CommandText = "SELECT city.name AS cityName, city.population
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.SetDataSource(myData)
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK)
End Try
```

C# Example

```
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;
```

```

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;
MySQL.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();
myAdapter = new MySQL.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.SetDataSource(myData);
    myViewer.ReportSource = myReport;
}
catch (MySQL.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

A new dataset is generated using the same query used to generate the previously saved dataset. Once the dataset is filled, a ReportDocument is used to load the report file and bind it to the dataset. The ReportDocument is then passed as the ReportSource of the crViewer.

This same approach is taken when a report is created from a single table using Connector/ODBC. The dataset replaces the table used in the report and the report is displayed properly.

When a report is created from multiple tables using Connector/ODBC, a dataset with multiple tables must be created in our application. This allows each table in the report data source to be replaced with a report in the dataset.

We populate a dataset with multiple tables by providing multiple SELECT statements in our MySqlCommand object. These SELECT statements are based on the SQL query shown in Crystal Reports in the Database menu's Show SQL Query option. Assume the following query:

```
SELECT `country`.`Name`, `country`.`Continent`, `country`.`Populatio
FROM `world`.`country` `country` LEFT OUTER JOIN `world`.`city` `cit
ORDER BY `country`.`Continent`, `country`.`Name`, `city`.`Name`
```

This query is converted to two SELECT queries and displayed with the following code:

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient
```

```
Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter
```

```
conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"
```

Try

```
conn.Open()
cmd.CommandText = "SELECT name, population, countrycode FROM cit
    & "SELECT name, population, code, continent FROM country ORD
cmd.Connection = conn
```

```
myAdapter.SelectCommand = cmd
myAdapter.Fill(myData)
```

```
myReport.Load(".\world_report.rpt")
myReport.Database.Tables(0).SetDataSource(myData.Tables(0))
myReport.Database.Tables(1).SetDataSource(myData.Tables(1))
myViewer.ReportSource = myReport
```

Catch ex As Exception

```
    MessageBox.Show(ex.Message, "Report could not be created", Messa
End Try
```

C# Example

```

using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT name, population, countrycode FROM cit
        "BY countrycode, name; SELECT name, population, code, contin
        "country ORDER BY continent, name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0));
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1));
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

It is important to order the SELECT queries in alphabetical order, as this is the order the report will expect its source tables to be in. One SetDataSource statement is needed for each table in the report.

This approach can cause performance problems because Crystal Reports must bind the tables together on the client-side, which will be slower than using a pre-saved dataset.

23.2.5.6. Handling Date and Time Information in Connector/NET

23.2.5.6.1. Introduction

MySQL and the .NET languages handle date and time information differently, with MySQL allowing dates that cannot be represented by a .NET data type, such as '0000-00-00 00:00:00'. These differences can cause problems if not properly handled.

In this section we will demonstrate how to properly handle date and time information when using Connector/NET.

23.2.5.6.2. Problems when Using Invalid Dates

The differences in date handling can cause problems for developers who use invalid dates. Invalid MySQL dates cannot be loaded into native .NET `DateTime` objects, including `NULL` dates.

Because of this issue, .NET `DataSet` objects cannot be populated by the `Fill` method of the `MySqlCommandAdapter` class as invalid dates will cause a `System.ArgumentOutOfRangeException` exception to occur.

23.2.5.6.3. Restricting Invalid Dates

The best solution to the date problem is to restrict users from entering invalid dates. This can be done on either the client or the server side.

Restricting invalid dates on the client side is as simple as always using the .NET `DateTime` class to handle dates. The `DateTime` class will only allow valid dates, ensuring that the values in your database are also valid. The disadvantage of this is that it is not useful in a mixed environment where .NET and non .NET code are used to manipulate the database, as each application must perform its own date validation.

Users of MySQL 5.0.2 and higher can use the new `traditional SQL` mode to restrict invalid date values. For information on using the `traditional SQL` mode, see [Section 5.2.5, “The Server SQL Mode”](#).

23.2.5.6.4. Handling Invalid Dates

Although it is strongly recommended that you avoid the use of invalid dates within your .NET application, it is possible to use invalid dates by means of the `MySqlDateTime` datatype.

The `MySqlDateTime` datatype supports the same date values that are supported by the MySQL server. The default behavior of Connector/NET is to return a .NET `DateTime` object for valid date values, and return an error for invalid dates. This default can be modified to cause Connector/NET to return `MySqlDateTime` objects for invalid dates.

To instruct Connector/NET to return a `MySqlDateTime` object for invalid dates, add the following line to your connection string:

```
Allow Zero Datetime=True
```

Please note that the use of the `MySqlDateTime` class can still be problematic. The following are some known issues:

1. Data binding for invalid dates can still cause errors (zero dates like 0000-00-00 do not seem to have this problem).
2. The `ToString` method return a date formatted in the standard MySQL format (for example, 2005-02-23 08:50:25). This differs from the `ToString` behavior of the .NET `DateTime` class.
3. The `MySqlDateTime` class supports NULL dates, while the .NET `DateTime` class does not. This can cause errors when trying to convert a `MySqlDateTime` to a `DateTime` if you do not check for NULL first.

Because of the known issues, the best recommendation is still to use only valid dates in your application.

23.2.5.6.5. Handling NULL Dates

The .NET `DateTime` datatype cannot handle NULL values. As such, when assigning values from a query to a `DateTime` variable, you must first check whether the value is in fact NULL.

When using a `MySqlDataReader`, use the `.IsDBNull` method to check whether a

value is NULL before making the assignment:

Visual Basic Example

```
If Not myReader.IsDBNull(myReader.GetOrdinal("mytime")) Then
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"))
Else
    myTime = DateTime.MinValue
End If
```

C# Example

```
if (! myReader.IsDBNull(myReader.GetOrdinal("mytime")))
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"));
else
    myTime = DateTime.MinValue;
```

NULL values will work in a dataset and can be bound to form controls without special handling.

23.2.6. Connector/NET Support

The developers of Connector/NET greatly value the input of our users in the software development process. If you find Connector/NET lacking some feature important to you, or if you discover a bug and need to file a bug report, please use the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

23.2.6.1. Connector/NET Community Support

- Community support for Connector/NET can be found through the forums at <http://forums.mysql.com>.
- Community support for Connector/NET can also be found through the mailing lists at <http://lists.mysql.com>.
- Paid support is available from MySQL AB. Additional information is available at <http://www.mysql.com/support/>.

23.2.6.2. How to report Connector/NET Problems or Bugs

If you encounter difficulties or problems with Connector/NET, contact the Connector/NET community [Section 23.2.6.1, “Connector/NET Community Support”](#).

You should first try to execute the same SQL statements and commands from the **mysql** client program or from admndemo. This helps you determine whether the error is in Connector/NET or MySQL.

If reporting a problem, you should ideally include the following information with the email:

- Operating system and version
- Connector/NET version
- MySQL server version
- Copies of error messages or other unexpected output
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through <http://bugs.mysql.com/>.

23.2.6.3. Connector/NET Change History

The Connector/NET Change History (Changelog) is located with the main Changelog for MySQL. See [Section D.4, “MySQL Connector/NET Change History”](#).

23.3. MySQL Connector/J

MySQL provides connectivity for client applications developed in the Java programming language via a JDBC driver, which is called MySQL Connector/J.

MySQL Connector/J is a JDBC-3.0 Type 4 driver, which means that is pure Java, implements version 3.0 of the JDBC specification, and communicates directly with the MySQL server using the MySQL protocol.

Although JDBC is useful by itself, we would hope that if you are not familiar with JDBC that after reading the first few sections of this manual, that you would avoid using naked JDBC for all but the most trivial problems and consider using one of the popular persistence frameworks such as [Hibernate](#), [Spring's JDBC templates](#) or [Ibatis SQL Maps](#) to do the majority of repetitive work and heavier lifting that is sometimes required with JDBC.

This section is not designed to be a complete JDBC tutorial. If you need more information about using JDBC you might be interested in the following online tutorials that are more in-depth than the information presented here:

- [JDBC Basics](#) — A tutorial from Sun covering beginner topics in JDBC
- [JDBC Short Course](#) — A more in-depth tutorial from Sun and JGuru

23.3.1. Connector/J Versions

There are currently three version of MySQL Connector/J available:

- Connector/J 3.0 provides core functionality and was designed with connectivity to MySQL 3.x or MySQL 4.1 servers, although it will provide basic compatibility with later versions of MySQL. Connector/J 3.0 does not support server-side prepared statements, and does not support any of the features in versions of MySQL later than 4.1.
- Connector/J 3.1 was designed for connectivity to MySQL 4.1 and MySQL 5.0 servers and provides support for all the functionality in MySQL 5.0 except distributed transaction (XA) support.

- Connector/J 5.0 provides support for all the functionality offered by Connector/J 3.1 and includes distributed transaction (XA) support.

The current recommended version for Connector/J is 5.0. This guide covers all three connector versions, with specific notes given where a setting applies to a specific option.

23.3.1.1. Java Versions Supported

MySQL Connector/J supports Java-2 JVMs, including:

- JDK 1.2.x
- JDK 1.3.x
- JDK 1.4.x
- JDK 1.5.x

If you are building Connector/J from source using the source distribution (see [Section 23.3.2.4, “Installing from the Development Source Tree”](#)) then you must use JDK 1.4.x or newer to compile the Connector package.

MySQL Connector/J does not support JDK-1.1.x or JDK-1.0.x

Because of the implementation of `java.sql.Savepoint`, Connector/J 3.1.0 and newer will not run on JDKs older than 1.4 unless the class verifier is turned off (by setting the `-Xverify:none` option to the Java runtime). This is because the class verifier will try to load the class definition for `java.sql.Savepoint` even though it is not accessed by the driver unless you actually use `savepoint` functionality.

Caching functionality provided by Connector/J 3.1.0 or newer is also not available on JVMs older than 1.4.x, as it relies on `java.util.LinkedHashMap` which was first available in JDK-1.4.0.

23.3.2. Installing Connector/J

You can install the Connector/J package using two methods, using either the

binary or source distribution. The binary distribution provides the easiest methods for installation; the source distribution enables you to customize your installation further. With either solution, you must

23.3.2.1. Installing Connector/J from a Binary Distribution

The easiest method of installation is to use the binary distribution of the Connector/J package. The binary distribution is available either as a Tar/Gzip or Zip file which you must extract to a suitable location and then optionally make the information about the package available by changing your CLASSPATH (see [Section 23.3.2.2, “Installing the Driver and Configuring the CLASSPATH”](#)).

MySQL Connector/J is distributed as a .zip or .tar.gz archive containing the sources, the class files, and the JAR archive named `mysql-connector-java-[version]-bin.jar`, and starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-[version]-bin-g.jar`.

Starting with Connector/J 3.1.9, the .class files that constitute the JAR files are only included as part of the driver JAR file.

You should not use the debug build of the driver unless instructed to do so when reporting a problem or bug to MySQL AB, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

You will need to use the appropriate graphical or command-line utility to unarchive the distribution (for example, WinZip for the .zip archive, and **tar** for the .tar.gz archive). Because there are potentially long filenames in the distribution, we use the GNU tar archive format. You will need to use GNU tar (or an application that understands the GNU tar archive format) to unpack the .tar.gz variant of the distribution.

23.3.2.2. Installing the Driver and Configuring the CLASSPATH

Once you have extracted the distribution archive, you can install the driver by placing `mysql-connector-java-[version]-bin.jar` in your classpath, either by adding the full path to it to your CLASSPATH environment variable, or by directly specifying it with the command line switch `-cp` when starting your JVM.

If you are going to use the driver with the JDBC DriverManager, you would use `com.mysql.jdbc.Driver` as the class that implements `java.sql.Driver`.

You can set the `CLASSPATH` environment variable under UNIX, Linux or Mac OS X either locally for a user within their `.profile`, `.login` or other login file. You can also set it globally by editing the global `/etc/profile` file.

For example, under a C shell (`csh`, `tcsh`) you would add the Connector/J driver to your `CLASSPATH` using the following:

```
shell> setenv CLASSPATH /path/to/mysql-connector-java-[version]-bin.
```

Or with a Bourne-compatible shell (`sh`, `ksh`, `bash`):

```
export set CLASSPATH=/path/to/mysql-connector-java-[version]-bin.jar
```

Within Windows 2000, Windows XP and Windows Server 2003, you must set the environment variable through the System control panel.

If you want to use MySQL Connector/J with an application server such as Tomcat or JBoss, you will have to read your vendor's documentation for more information on how to configure third-party class libraries, as most application servers ignore the `CLASSPATH` environment variable. For configuration examples for some J2EE application servers, see [Section 23.3.5.2, “Using Connector/J with J2EE and Other Java Frameworks”](#). However, the authoritative source for JDBC connection pool configuration information for your particular application server is the documentation for that application server.

If you are developing servlets or JSPs, and your application server is J2EE-compliant, you can put the driver's `.jar` file in the `WEB-INF/lib` subdirectory of your webapp, as this is a standard location for third party class libraries in J2EE web applications.

You can also use the `MysqlDataSource` or `MysqlConnectionPoolDataSource` classes in the `com.mysql.jdbc.jdbc2.optional` package, if your J2EE application server supports or requires them. Starting with Connector/J 5.0.0, the `javax.sql.XADataSource` interface is implemented via the `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class, which supports XA distributed transactions when used in combination with MySQL server version 5.0.

The various `MysqlDataSource` classes support the following parameters (through standard set mutators):

- `user`
- `password`
- `serverName` (see the previous section about fail-over hosts)
- `databaseName`
- `port`

23.3.2.3. Upgrading from an Older Version

MySQL AB tries to keep the upgrade process as easy as possible, however as is the case with any software, sometimes changes need to be made in new versions to support new features, improve existing functionality, or comply with new standards.

This section has information about what users who are upgrading from one version of Connector/J to another (or to a new version of the MySQL server, with respect to JDBC functionality) should be aware of.

23.3.2.3.1. Upgrading from MySQL Connector/J 3.0 to 3.1

Connector/J 3.1 is designed to be backward-compatible with Connector/J 3.0 as much as possible. Major changes are isolated to new functionality exposed in MySQL-4.1 and newer, which includes Unicode character sets, server-side prepared statements, `SQLState` codes returned in error messages by the server and various performance enhancements that can be enabled or disabled via configuration properties.

- **Unicode Character Sets** — See the next section, as well as [Chapter 10, Character Set Support](#), for information on this new feature of MySQL. If you have something misconfigured, it will usually show up as an error with a message similar to `Illegal mix of collations`.
- **Server-side Prepared Statements** — Connector/J 3.1 will automatically

detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer).

Starting with version 3.1.7, the driver scans SQL you are preparing via all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this feature by passing `emulateUnsupportedPstmts=false` in your JDBC URL.

If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the connection property `useServerPrepStmts=false`

- **Datetimes** with all-zero components (`0000-00-00 . . .`) — These values can not be represented reliably in Java. Connector/J 3.0.x always converted them to NULL when being read from a `ResultSet`.

Connector/J 3.1 throws an exception by default when these values are encountered as this is the most correct behavior according to the JDBC and SQL standards. This behavior can be modified using the `zeroDateTimeBehavior` configuration property. The allowable values are:

- `exception` (the default), which throws an `SQLException` with an `SQLState` of `S1009`.
- `convertToNull`, which returns NULL instead of the date.
- `round`, which rounds the date to the nearest closest value which is `0001-01-01`.

Starting with Connector/J 3.1.7, `ResultSet.getString()` can be decoupled from this behavior via `noDatetimeStringSync=true` (the default value is `false`) so that you can get retrieve the unaltered all-zero value as a `String`. It should be noted that this also precludes using any time zone conversions, therefore the driver will not allow you to enable `noDatetimeStringSync` and `useTimezone` at the same time.

- **New SQLState Codes** — Connector/J 3.1 uses SQL:1999 SQLState codes

returned by the MySQL server (if supported), which are different from the legacy X/Open state codes that Connector/J 3.0 uses. If connected to a MySQL server older than MySQL-4.1.0 (the oldest version to return SQLStates as part of the error code), the driver will use a built-in mapping. You can revert to the old mapping by using the configuration property `useSqlStateCodes=false`.

- **ResultSet.getString()** — Calling `ResultSet.getString()` on a BLOB column will now return the address of the `byte[]` array that represents it, instead of a `String` representation of the BLOB. BLOBs have no character set, so they can't be converted to `java.lang.Strings` without data loss or corruption.

To store strings in MySQL with LOB behavior, use one of the TEXT types, which the driver will treat as a `java.sql.Clob`.

- **Debug builds** — Starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-[version]-bin-g.jar` is shipped alongside the normal binary jar file that is named `mysql-connector-java-[version]-bin.jar`.

Starting with Connector/J 3.1.9, we don't ship the `.class` files unbundled, they are only available in the JAR archives that ship with the driver.

You should not use the debug build of the driver unless instructed to do so when reporting a problem or bug to MySQL AB, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

23.3.2.3.2. JDBC-Specific Issues When Upgrading to MySQL Server 4.1 or Newer

- *Using the UTF-8 Character Encoding* - Prior to MySQL server version 4.1, the UTF-8 character encoding was not supported by the server, however the JDBC driver could use it, allowing storage of multiple character sets in latin1 tables on the server.

Starting with MySQL-4.1, this functionality is deprecated. If you have

applications that rely on this functionality, and can not upgrade them to use the official Unicode character support in MySQL server version 4.1 or newer, you should add the following property to your connection URL:

```
useOldUTF8Behavior=true
```

- *Server-side Prepared Statements* - Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer). If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the following connection property:

```
useServerPrepStmts=false
```

23.3.2.4. Installing from the Development Source Tree

Caution. You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL Connector/J up and running on your system, you should use a standard release distribution.

To install MySQL Connector/J from the development source tree, make sure that you have the following prerequisites:

- Subversion, to check out the sources from our repository (available from <http://subversion.tigris.org/>).
- Apache Ant version 1.6 or newer (available from <http://ant.apache.org/>).
- JDK-1.4.2 or later. Although MySQL Connector/J can be installed on older JDKs, to compile it from source you must have at least JDK-1.4.2.

The Subversion source code repository for MySQL Connector/J is located at <http://svn.mysql.com/svnpublic/connector-j>. In general, you should not check out the entire repository because it contains every branch and tag for MySQL Connector/J and is quite large.

To check out and compile a specific branch of MySQL Connector/J, follow these steps:

1. At the time of this writing, there are three active branches of Connector/J: `branch_3_0`, `branch_3_1` and `branch_5_0`. Check out the latest code from the branch that you want with the following command (replacing `[major]` and `[minor]` with appropriate version numbers):

```
shell> svn co »  
http://svn.mysql.com/svnpublic/connector-j/branches/branch_[major]
```

This creates a `connector-j` subdirectory in the current directory that contains the latest sources for the requested branch.

2. Change location to the `connector-j` directory to make it your current working directory:

```
shell> cd connector-j
```

3. Issue the following command to compile the driver and create a `.jar` file suitable for installation:

```
shell> ant dist
```

This creates a `build` directory in the current directory, where all build output will go. A directory is created in the `build` directory that includes the version number of the sources you are building from. This directory contains the sources, compiled `.class` files, and a `.jar` file suitable for deployment. For other possible targets, including ones that will create a fully packaged distribution, issue the following command:

```
shell> ant --projecthelp
```

4. A newly created `.jar` file containing the JDBC driver will be placed in the directory `build/mysql-connector-java-[version]`.

Install the newly created JDBC driver as you would a binary `.jar` file that you download from MySQL by following the instructions in [Section 23.3.2.2, “Installing the Driver and Configuring the CLASSPATH”](#).

23.3.3. Connector/J Examples

Examples of using Connector/J are located throughout this document, this section provides a summary and links to these examples.

- [Example 23.1, “Obtaining a connection from the DriverManager”](#)
- [Example 23.2, “Using java.sql.Statement to execute a SELECT query”](#)
- [Example 23.3, “Stored Procedures”](#)
- [Example 23.4, “Using Connection.prepareStatement\(\)”](#)
- [Example 23.5, “Registering output parameters”](#)
- [Example 23.6, “Setting CallableStatement input parameters”](#)
- [Example 23.7, “Retrieving results and output parameter values”](#)
- [Example 23.8, “Retrieving AUTO INCREMENT column values using Statement.getGeneratedKeys\(\)”](#)
- [Example 23.9, “Retrieving AUTO INCREMENT column values using SELECT LAST_INSERT_ID\(\)”](#)
- [Example 23.10, “Retrieving AUTO INCREMENT column values in Updatable ResultSets”](#)
- [Example 23.11, “Using a connection pool with a J2EE application server”](#)
- [Example 23.12, “Example of transaction with retry logic”](#)

23.3.4. Connector/J (JDBC) Reference

This section of the manual contains reference material for MySQL Connector/J, some of which is automatically generated during the Connector/J build process.

23.3.4.1. Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J

The name of the class that implements `java.sql.Driver` in MySQL Connector/J is `com.mysql.jdbc.Driver`. The `org.gjt.mm.mysql.Driver` class name is also usable to remain backward-compatible with MM.MySQL. You should use this class name when registering the driver, or when otherwise configuring software to use MySQL Connector/J.

The JDBC URL format for MySQL Connector/J is as follows, with items in square brackets ([,]) being optional:

```
jdbc:mysql://[host][, failoverhost...][:port]/[database] »  
[?propertyName1]=[propertyValue1]&[propertyName2]=[propertyValue2]..
```

If the hostname is not specified, it defaults to 127.0.0.1. If the port is not specified, it defaults to 3306, the default port number for MySQL servers.

```
jdbc:mysql://[host:port],[host:port].../[database] »  
[?propertyName1]=[propertyValue1]&[propertyName2]=[propertyValue2]..
```

If the database is not specified, the connection will be made with no default database. In this case, you will need to either call the `setCatalog()` method on the Connection instance or fully-specify table names using the database name (i.e. `SELECT dbname.tablename.colname FROM dbname.tablename...`) in your SQL. Not specifying the database to use upon connection is generally only useful when building tools that work with multiple databases, such as GUI database managers.

MySQL Connector/J has fail-over support. This allows the driver to fail-over to any number of slave hosts and still perform read-only queries. Fail-over only happens when the connection is in an `autoCommit(true)` state, because fail-over can not happen reliably when a transaction is in progress. Most application servers and connection pools set `autoCommit` to `true` at the end of every transaction/connection use.

The fail-over functionality has the following behavior:

- If the URL property `autoReconnect` is `false`: Failover only happens at connection initialization, and failback occurs when the driver determines that the first host has become available again.
- If the URL property `autoReconnect` is `true`: Failover happens when the driver determines that the connection has failed (before *every* query), and falls back to the first host when it determines that the host has become available again (after `queriesBeforeRetryMaster` queries have been issued).

In either case, whenever you are connected to a "failed-over" server, the connection will be set to read-only state, so queries that would modify data will

have exceptions thrown (the query will **never** be processed by the MySQL server).

Configuration properties define how Connector/J will make a connection to a MySQL server. Unless otherwise noted, properties can be set for a DataSource object or for a Connection object.

Configuration Properties can be set in one of the following ways:

- Using the set*() methods on MySQL implementations of java.sql.DataSource (which is the preferred method when using implementations of java.sql.DataSource):
 - com.mysql.jdbc.jdbc2.optional.MysqlDataSource
 - com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
- As a key/value pair in the java.util.Properties instance passed to DriverManager.getConnection() or Driver.connect()
- As a JDBC URL parameter in the URL given to java.sql.DriverManager.getConnection(), java.sql.Driver.connect() or the MySQL implementations of the javax.sql.DataSource setURL() method.

Note. If the mechanism you use to configure a JDBC URL is XML-based, you will need to use the XML character literal &#amp; to separate configuration parameters, as the ampersand is a reserved character for XML.

The properties are listed in the following tables.

Connection/Authentication.

Property Name	Definition	Default
user	The user to connect as	
password	The password to use when connecting	
	The name of the class that the driver should use for	

socketFactory	creating socket connections to the server. This class must implement the interface <code>com.mysql.jdbc.SocketFactory</code> and have public no-args constructor.	com.r
connectTimeout	Timeout for socket connect (in milliseconds), with 0 being no timeout. Only works on JDK-1.4 or newer. Defaults to 0.	0
socketTimeout	Timeout on network socket operations (0, the default means no timeout).	0
useConfigs	Load the comma-delimited list of configuration properties before parsing the URL or applying user-specified properties. See Section 23.3.4.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”	
interactiveClient	Set the <code>CLIENT_INTERACTIVE</code> flag, which tells MySQL to timeout connections based on <code>INTERACTIVE_TIMEOUT</code> instead of <code>WAIT_TIMEOUT</code>	false
propertiesTransform	An implementation of <code>com.mysql.jdbc.ConnectionPropertiesTransform</code> that the driver will use to modify URL properties passed to the driver before attempting a connection	
useCompression	Use zlib compression when communicating with the server (true/false)? Defaults to false.	false

High Availability and Clustering.

Property Name	Definition	Default Value	Since Version
	Should the driver try to re-establish stale and/or dead connections? If enabled the driver will throw an exception for a queries issued on a stale or dead connection, which belong to the current transaction, but will attempt reconnect before the next		

autoReconnect	query issued on the connection in a new transaction. The use of this feature is not recommended, because it has side effects related to session state and data consistency when applications don'thandle SQLExceptions properly, and is only designed to be used when you are unable to configure your application to handle SQLExceptions resulting from dead andstale connections properly. Alternatively, investigate setting the MySQL server variable "wait_timeout"to some high value rather than the default of 8 hours.	false	1.1
autoReconnectForPools	Use a reconnection strategy appropriate for connection pools (defaults to 'false')	false	3.1.3
failOverReadOnly	When failing over in autoReconnect mode, should the connection be set to 'read-only'?	true	3.0.12
reconnectAtTxEnd	If autoReconnect is set to true, should the driver attempt reconnectionsat the end of every transaction?	false	3.0.10
roundRobinLoadBalance	When autoReconnect is enabled, and failoverReadonly is false, should we pick hosts to connect to on a round-robin basis?	false	3.1.2
queriesBeforeRetryMaster	Number of queries to issue before falling back to master when failed over (when using multi-host failover). Whichever condition is met first, 'queriesBeforeRetryMaster' or	50	3.0.2

	'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Defaults to 50.		
secondsBeforeRetryMaster	How long should the driver wait, when failed over, before attempting to reconnect to the master server? Whichever condition is met first, 'queriesBeforeRetryMaster' or 'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Time in seconds, defaults to 30	30	3.0.2
resourceId	A globally unique name that identifies the resource that this datasource or connection is connected to, used for <code>XAResource.isSameRM()</code> when the driver can't determine this value based on hostnames used in the URL		5.0.1

Security.

Property Name	Definition	Default Value	Since Version
allowMultiQueries	Allow the use of ';' to delimit multiple queries during one statement (true/false, defaults to 'false')	false	3.1.1
useSSL	Use SSL when communicating with the server (true/false), defaults to 'false'	false	3.0.2
requireSSL	Require SSL connection if useSSL=true? (defaults to 'false').	false	3.1.0
allowUrlInLocalInfile	Should the driver allow URLs in 'LOAD DATA LOCAL INFILE' statements?	false	3.1.4

paranoid	Take measures to prevent exposure sensitive information in error messages and clear data structures holding sensitive data when possible? (defaults to 'false')	false	3.0.1
----------	---	-------	-------

Performance Extensions.

Property Name	Definition
metadataCacheSize	The number of queries to cacheResultSetMetadata for if cacheResultSetMetaData is set to 'true' (default 50)
prepStmtCacheSize	If prepared statement caching is enabled, how many prepared statements should be cached?
prepStmtCacheSqlLimit	If prepared statement caching is enabled, what's the largest SQL the driver will cache the parsing for?
useCursorFetch	If connected to MySQL > 5.0.2, and setFetchSize() > 0 on a statement, should that statement use cursor-based fetching to retrieve rows?
blobSendChunkSize	Chunk to use when sending BLOB/CLOBs via ServerPreparedStatements
cacheCallableStmts	Should the driver cache the parsing stage of CallableStatements
cachePrepStmts	Should the driver cache the parsing stage of PreparedStatements of client-side prepared statements, the check for suitability of server-side prepared and server-side prepared statements themselves?
	Should the driver cache

cacheResultSetMetadata	ResultSetMetaData for Statements and PreparedStatements? (Req. JDK-1.4+, true/false, default 'false')
cacheServerConfiguration	Should the driver cache the results of SHOW VARIABLES and SHOW COLLATION on a per-URL basis?
defaultFetchSize	The driver will call setFetchSize(n) with this value on all newly-created Statements
dontTrackOpenResources	The JDBC specification requires the driver to automatically track and close resources, however if your application doesn't do a good job of explicitly calling close() on statements or result sets, this can cause memory leakage. Setting this property to true relaxes this constraint, and can be more memory efficient for some applications.
dynamicCalendars	Should the driver retrieve the default calendar when required, or cache it per connection/session?
elideSetAutoCommits	If using MySQL-4.1 or newer, should the driver only issue 'set autocommit=n' queries when the server's state doesn't match the requested state by Connection.setAutoCommit(boolean)?
holdResultsOpenOverStatementClose	Should the driver close result sets on Statement.close() as required by the JDBC specification?
locatorFetchBufferSize	If 'emulateLocators' is configured to 'true', what size buffer should be used when fetching BLOB data for getBinaryInputStream?
	Should the driver use multiqueries (irregardless of the setting of allowMultiQueries) as well as rewriting of prepared statements for

rewriteBatchedStatements	<p>INSERT into multi-value inserts when executeBatch() is called? Notice that this has the potential for SQL injection if using plain java.sql.Statements and your code doesn't sanitize input correctly. Notice that for prepared statements, server-side prepared statements can not currently take advantage of this rewrite option, and that if you don't specify stream lengths when using PreparedStatement.set*Stream(), the driver won't be able to determine the optimum number of parameters per batch and you might receive an error from the driver that the resultant packet is too large.</p> <p>Statement.getGeneratedKeys() for these rewritten statements only works when the entire batch includes INSERT statements.</p>
useFastIntParsing	Use internal String->Integer conversion routines to avoid excessive object creation?
useJvmCharsetConverters	Always use the character encoding routines built into the JVM, rather than using lookup tables for single-byte character sets? (The default of "true" for this is appropriate for newer JVMs)
useLocalSessionState	Should the driver refer to the internal values of autocommit and transaction isolation that are set by Connection.setAutoCommit() and Connection.setTransactionIsolation(), rather than querying the database?
useReadAheadInput	Use newer, optimized non-blocking, buffered input stream when reading from the server?

Debugging/Profiling.

Property Name	Definition	Defa
logger	The name of a class that implements 'com.mysql.jdbc.log.Log' that will be used to log messages to.(default is 'com.mysql.jdbc.log.StandardLogger', which logs to STDERR)	com.
profileSQL	Trace queries and their execution/fetch times to the configured logger (true/false) defaults to 'false'	false
reportMetricsIntervalMillis	If 'gatherPerfMetrics' is enabled, how often should they be logged (in ms)?	3000
maxQuerySizeToLog	Controls the maximum length/size of a query that will get logged when profiling or tracing	2048
packetDebugBufferSize	The maximum number of packets to retain when 'enablePacketDebug' is true	20
slowQueryThresholdMillis	If 'logSlowQueries' is enabled, how long should a query (in ms) before it is logged as 'slow'?	2000
useUsageAdvisor	Should the driver issue 'usage' warnings advising proper and efficient usage of JDBC and MySQL Connector/J to the log (true/false, defaults to 'false')?	false
autoGenerateTestcaseScript	Should the driver dump the SQL it is executing, including server-side prepared statements to STDERR?	false
dumpMetadataOnColumnNotFound	Should the driver dump the field-level metadata of a result set into the exception message when	false

	ResultSet.findColumn() fails?	
dumpQueriesOnException	Should the driver dump the contents of the query sent to the server in the message for SQLExceptions?	false
enablePacketDebug	When enabled, a ring-buffer of 'packetDebugBufferSize' packets will be kept, and dumped when exceptions are thrown in key areas in the driver's code	false
explainSlowQueries	If 'logSlowQueries' is enabled, should the driver automatically issue an 'EXPLAIN' on the server and send the results to the configured log at a WARN level?	false
logSlowQueries	Should queries that take longer than 'slowQueryThresholdMillis' be logged?	false
traceProtocol	Should trace-level network protocol be logged?	false

Miscellaneous.

Property Name	Definition
useUnicode	Should the driver use Unicode character strings? Should only be used when you are using a character set mapping, or you are using a character set that MySQL either supports (UTF-8), true/false, defaults to 'true'
characterEncoding	If 'useUnicode' is set to true, what character encoding should the driver use when dealing with strings?
characterSetResults	Character set to tell the server to use for the results
connectionCollation	If set, tells the server to use this collation for the connection
sessionVariables	A comma-separated list of name=value pairs to be sent to the server when the connection is established

allowNanAndInf	Should the driver allow NaN or - PreparedStatement.setDouble()?
autoClosePStmtStreams	Should the driver automatically c passed as arguments via set*() m
autoDeserialize	Should the driver automatically c in BLOB fields?
capitalizeTypeNames	Capitalize type names in Databases when using WebObjects, true/false
clobCharacterEncoding	The character encoding to use fo MEDIUMTEXT and LONGTEXT connection characterEncoding
clobberStreamingResults	This will cause a 'streaming' Res and any outstanding data still str discarded if another query is exe read from the server.
continueBatchOnError	Should the driver continue proce statement fails. The JDBC spec a
createDatabaseIfNotExist	Creates the database given in the Assumes the configured user has
emptyStringsConvertToZero	Should the driver allow conversi numeric values of '0'?
emulateLocators	N/A
emulateUnsupportedPstmts	Should the driver detect prepared by the server, and replace them v
ignoreNonTxTables	Ignore non-transactional table w 'false').
jdbcCompliantTruncation	Should the driver throw java.sql. data is truncated as is required by connected to a server that suppor newer)?
maxRows	The maximum number of rows to all rows).
noAccessToProcedureBodies	When determining procedure par CallableStatements, and the conr bodies through "SHOW CREATI

	mysql.proc should the driver inst parameters reported as INOUT V exception?
noDatetimeStringSync	Don't ensure that ResultSet.getDatetimeType().toS
noTimezoneConversionForTimeType	Don't convert TIME values using 'useTimezone'='true'
nullCatalogMeansCurrent	When DatabaseMetadataMethod the value null mean use the curre compliant, but follows legacy be (driver)
nullNamePatternMatchesAll	Should DatabaseMetaData metho treat null the same as '%' (this is versions of the driver accepted th
overrideSupportsIntegrityEnhancementFacility	Should the driver return "true" fo DatabaseMetaData.supportsInteg the database doesn't support it to require this method to return "tru even though the SQL specificatio much more than just foreign key (OpenOffice)?
pedantic	Follow the JDBC spec to the lett
pinGlobalTxToPhysicalConnection	When using XAConnections, sho operations on a given XID are al connection? This allows the XA(... JOIN" after "XA END" has be
processEscapeCodesForPrepStmts	Should the driver process escape
relaxAutoCommit	If the version of MySQL the driv transactions, still allow calls to c setAutoCommit() (true/false, def
retainStatementAfterResultSetClose	Should the driver retain the State ResultSet.close() has been called JDBC-4.0.
rollbackOnPooledClose	Should the driver issue a rollback pool is closed?
	Enables workarounds for bugs in

runningCTS13	version 1.3
serverTimezone	Override detection/mapping of time zone if the server doesn't map to Java time zone
strictFloatingPoint	Used only in older versions of connector
strictUpdates	Should the driver do strict checking of updatable result sets (true, false, or ignore)
tinyInt1isBit	Should the driver treat the datatype TINYINT(1) as BIT (because the server silently converts TINYINT(1) to BIT when creating tables)?
transformedBitIsBoolean	If the driver converts TINYINT(1) to BOOLEAN instead of BIT for full compatibility with MySQL-5.0, as MySQL-5.0 has a BIT type
ultraDevHack	Create PreparedStatements for prepared statements. UltraDev is broken and issues a java.sql.SQLException (true/false, defaults to 'false')
useGmtMillisForDatetimes	Convert between session timezone and Timestamp instances (value true leads to more JDBC-compliant behavior)
useHostsInPrivileges	Add '@hostname' to users in DatabaseMetaData.getColumnPrivilegesForTable() defaults to 'true'.
useInformationSchema	When connected to MySQL-5.0. Use INFORMATION_SCHEMA to calculate column counts in DatabaseMetaData?
useJDBCCompliantTimezoneShift	Should the driver use JDBC-compliant TIME/TIMESTAMP/DATETIME conversions for those JDBC arguments which take time values (Notice that this option is exclusive with useServerPrepStmts configuration option.)
useOldUTF8Behavior	Use the UTF-8 behavior the driver had in 4.0 and older servers
useOnlyServerErrorMessages	Don't prepend 'standard' SQLState codes to error messages returned by the server.
useServerPrepStmts	Use server-side prepared statements (defaults to 'true').

<code>useSqlStateCodes</code>	Use SQL Standard state codes in codes (true/false), default is 'true'
<code>useStreamLengthsInPrepStmts</code>	Honor stream length parameter in PreparedStatement/ResultSet.set (true/false, defaults to 'true')?
<code>useTimezone</code>	Convert time/date types between (true/false, defaults to 'false')?
<code>useUnbufferedInput</code>	Don't use BufferedInputStream f
<code>yearIsDateType</code>	Should the JDBC driver treat the java.sql.Date, or as a SHORT?
<code>zeroDateTimeBehavior</code>	What should happen when the dr that are composed entirely of zer invalid dates)? Valid values are 'e 'convertToNull'.

Connector/J also supports access to MySQL via named pipes on Windows NT/2000/XP using the `NamedPipeSocketFactory` as a plugin-socket factory via the `socketFactory` property. If you don't use a `namedPipePath` property, the default of `\\.\pipe\MySQL` will be used. If you use the `NamedPipeSocketFactory`, the hostname and port number values in the JDBC url will be ignored. You can enable this feature using:

```
socketFactory=com.mysql.jdbc.NamedPipeSocketFactory
```

Named pipes only work when connecting to a MySQL server on the same physical machine as the one the JDBC driver is being used on. In simple performance tests, it appears that named pipe access is between 30%-50% faster than the standard TCP/IP access.

You can create your own socket factories by following the example code in `com.mysql.jdbc.NamedPipeSocketFactory`, or `com.mysql.jdbc.StandardSocketFactory`.

23.3.4.2. JDBC API Implementation Notes

MySQL Connector/J passes all of the tests in the publicly-available version of Sun's JDBC compliance test suite. However, in many places the JDBC

specification is vague about how certain functionality should be implemented, or the specification allows leeway in implementation.

This section gives details on a interface-by-interface level about how certain implementation decisions may affect how you use MySQL Connector/J.

- Blob

The Blob implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, you should use the corresponding `PreparedStatement.setBlob()` or `ResultSet.updateBlob()` (in the case of updatable result sets) methods to save changes back to the database.

Starting with Connector/J version 3.1.0, you can emulate Blobs with locators by adding the property 'emulateLocators=true' to your JDBC URL. You must then use a column alias with the value of the column set to the actual name of the Blob column in the SELECT that you write to retrieve the Blob. The SELECT must also reference only one table, the table must have a primary key, and the SELECT must cover all columns that make up the primary key. The driver will then delay loading the actual Blob data until you retrieve the Blob and call retrieval methods (`getInputStream()`, `getBytes()`, and so forth) on it.

- CallableStatement

Starting with Connector/J 3.1.1, stored procedures are supported when connecting to MySQL version 5.0 or newer via the `CallableStatement` interface. Currently, the `getParameterMetaData()` method of `CallableStatement` is not supported.

- Clob

The Clob implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, you should use the `PreparedStatement.setClob()` method to save changes back to the database. The JDBC API does not have a `ResultSet.updateClob()` method.

- Connection

Unlike older versions of MM.MySQL the `isClosed()` method does not ping the server to determine if it is alive. In accordance with the JDBC specification, it only returns true if `closed()` has been called on the connection. If you need to determine if the connection is still valid, you should issue a simple query, such as `SELECT 1`. The driver will throw an exception if the connection is no longer valid.

- DatabaseMetaData

Foreign Key information (`getImportedKeys()/getExportedKeys()` and `getCrossReference()`) is only available from InnoDB tables. However, the driver uses `SHOW CREATE TABLE` to retrieve this information, so when other storage engines support foreign keys, the driver will transparently support them as well.

- PreparedStatement

PreparedStatements are implemented by the driver, as MySQL does not have a prepared statement feature. Because of this, the driver does not implement `getParameterMetaData()` or `getMetaData()` as it would require the driver to have a complete SQL parser in the client.

Starting with version 3.1.0 MySQL Connector/J, server-side prepared statements and binary-encoded result sets are used when the server supports them.

Take care when using a server-side prepared statement with **large** parameters that are set via `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()`, or `setClob()`. If you want to re-execute the statement with any large parameter changed to a non-large parameter, it is necessary to call `clearParameters()` and set all parameters again. The reason for this is as follows:

- The driver streams the large data out-of-band to the prepared statement on the server side when the parameter is set (before execution of the prepared statement).
- Once that has been done, the stream used to read the data on the client

side is closed (as per the JDBC spec), and can't be read from again.

- If a parameter changes from large to non-large, the driver must reset the server-side state of the prepared statement to allow the parameter that is being changed to take the place of the prior large value. This removes all of the large data that has already been sent to the server, thus requiring the data to be re-sent, via the `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()` or `setClob()` methods.

Consequently, if you want to change the type of a parameter to a non-large one, you must call `clearParameters()` and set all parameters of the prepared statement again before it can be re-executed.

- **ResultSet**

By default, ResultSets are completely retrieved and stored in memory. In most cases this is the most efficient way to operate, and due to the design of the MySQL network protocol is easier to implement. If you are working with ResultSets that have a large number of rows or large values, and can not allocate heap space in your JVM for the memory required, you can tell the driver to stream the results back one row at a time.

To enable this functionality, you need to create a Statement instance in the following manner:

```
stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY
                             java.sql.ResultSet.CONCUR_READ_ONLY);
stmt.setFetchSize(Integer.MIN_VALUE);
```

The combination of a forward-only, read-only result set, with a fetch size of `Integer.MIN_VALUE` serves as a signal to the driver to stream result sets row-by-row. After this any result sets created with the statement will be retrieved row-by-row.

There are some caveats with this approach. You will have to read all of the rows in the result set (or close it) before you can issue any other queries on the connection, or an exception will be thrown.

The earliest the locks these statements hold can be released (whether they

be MyISAM table-level locks or row-level locks in some other storage engine such as InnoDB) is when the statement completes.

If the statement is within scope of a transaction, then locks are released when the transaction completes (which implies that the statement needs to complete first). As with most other databases, statements are not complete until all the results pending on the statement are read or the active result set for the statement is closed.

Therefore, if using streaming results, you should process them as quickly as possible if you want to maintain concurrent access to the tables referenced by the statement producing the result set.

- **ResultSetMetaData**

The `isAutoIncrement()` method only works when using MySQL servers 4.0 and newer.

- **Statement**

When using versions of the JDBC driver earlier than 3.2.1, and connected to server versions earlier than 5.0.3, the `setFetchSize()` method has no effect, other than to toggle result set streaming as described above.

MySQL does not support SQL cursors, and the JDBC driver doesn't emulate them, so `setCursorName()` has no effect.

23.3.4.3. Java, JDBC and MySQL Types

MySQL Connector/J is flexible in the way it handles conversions between MySQL data types and Java data types.

In general, any MySQL data type can be converted to a `java.lang.String`, and any numerical type can be converted to any of the Java numerical types, although round-off, overflow, or loss of precision may occur.

Starting with Connector/J 3.1.0, the JDBC driver will issue warnings or throw `DataTruncation` exceptions as is required by the JDBC specification unless the connection was configured not to do so by using the property

jdbcCompliantTruncation and setting it to false.

The conversions that are always guaranteed to work are listed in the following table:

Connection Properties - Miscellaneous.

These MySQL Data Types	Can always be converted to these Java types
CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET	java.lang.String, java.io.InputStream, java.io.Reader, java.sql.Blob, java.sql.Clob
FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT	java.lang.String, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Double, java.math.BigDecimal
DATE, TIME, DATETIME, TIMESTAMP	java.lang.String, java.sql.Date, java.sql.Timestamp

Note: round-off, overflow or loss of precision may occur if you choose a Java numeric data type that has less precision or capacity than the MySQL data type you are converting to/from.

The `ResultSet.getObject()` method uses the following type conversions between MySQL and Java types, following the JDBC specification where appropriate:

MySQL Types to Java Types for `ResultSet.getObject()`.

MySQL Type Name	Returned as Java Class
BIT(1) (new in MySQL-5.0)	java.lang.Boolean
BIT(> 1) (new in MySQL-5.0)	byte[]
TINYINT	java.lang.Boolean if the configuration property <code>tinyInt1isBit</code> is set to true (the default) and the storage size is 1, or java.lang.Integer if not.
	See TINYINT, above as these are aliases for

BOOL, BOOLEAN	TINYINT(1), currently.
SMALLINT[(M)] [UNSIGNED]	java.lang.Integer (regardless if UNSIGNED or not)
MEDIUMINT[(M)] [UNSIGNED]	java.lang.Integer, if UNSIGNED java.lang.Long
INT,INTEGER[(M)] [UNSIGNED]	java.lang.Integer, if UNSIGNED java.lang.Long
BIGINT[(M)] [UNSIGNED]	java.lang.Long, if UNSIGNED java.math.BigInteger
FLOAT[(M,D)]	java.lang.Float
DOUBLE[(M,B)]	java.lang.Double
DECIMAL[(M[,D])]	java.math.BigDecimal
DATE	java.sql.Date
DATETIME	java.sql.Timestamp
TIMESTAMP[(M)]	java.sql.Timestamp
TIME	java.sql.Time
YEAR[(2 4)]	java.sql.Date (with the date set two January 1st, at midnight)
CHAR(M)	java.lang.String (unless the character set for the column is BINARY, then byte[] is returned.
VARCHAR(M) [BINARY]	java.lang.String (unless the character set for the column is BINARY, then byte[] is returned.
BINARY(M)	byte[]
VARBINARY(M)	byte[]
TINYBLOB	byte[]
TINYTEXT	java.lang.String
BLOB	byte[]
TEXT	java.lang.String
MEDIUMBLOB	byte[]
MEDIUMTEXT	java.lang.String
LOBLOB	byte[]
LONGTEXT	java.lang.String

ENUM('value1','value2',...)	java.lang.String
SET('value1','value2',...)	java.lang.String

23.3.4.4. Using Character Sets and Unicode

All strings sent from the JDBC driver to the server are converted automatically from native Java Unicode form to the client character encoding, including all queries sent via `Statement.execute()`, `Statement.executeUpdate()`, `Statement.executeQuery()` as well as all `PreparedStatement` and `CallableStatement` parameters with the exclusion of parameters set using `setBytes()`, `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()` and `setBlob()`.

Prior to MySQL Server 4.1, Connector/J supported a single character encoding per connection, which could either be automatically detected from the server configuration, or could be configured by the user through the `useUnicode` and `"characterEncoding"` properties.

Starting with MySQL Server 4.1, Connector/J supports a single character encoding between client and server, and any number of character encodings for data returned by the server to the client in `ResultSets`.

The character encoding between client and server is automatically detected upon connection. The encoding used by the driver is specified on the server via the `character_set` system variable for server versions older than 4.1.0 and `character_set_server` for server versions 4.1.0 and newer. For more information, see [Section 10.3.1, “Server Character Set and Collation”](#).

To override the automatically-detected encoding on the client side, use the `characterEncoding` property in the URL used to connect to the server.

When specifying character encodings on the client side, Java-style names should be used. The following table lists Java-style names for MySQL character sets:

MySQL to Java Encoding Name Translations.

MySQL Character Set Name	Java-Style Character Encoding Name

usa7	US-ASCII
big5	Big5
gbk	GBK
sjis	SJIS (or Cp932 or MS932 for MySQL Server < 4.1.11)
cp932	Cp932 or MS932 (MySQL Server > 4.1.11)
gb2312	EUC_CN
ujis	EUC_JP
euc_kr	EUC_KR
latin1	ISO8859_1
latin1_de	ISO8859_1
german1	ISO8859_1
danish	ISO8859_1
latin2	ISO8859_2
czech	ISO8859_2
hungarian	ISO8859_2
croat	ISO8859_2
greek	ISO8859_7
hebrew	ISO8859_8
latin5	ISO8859_9
latvian	ISO8859_13
latvian1	ISO8859_13
estonia	ISO8859_13
dos	Cp437
pclatin2	Cp852
cp866	Cp866
koi8_ru	KOI8_R
tis620	TIS620
win1250	Cp1250
win1250ch	Cp1250
win1251	Cp1251

cp1251	Cp1251
win1251ukr	Cp1251
cp1257	Cp1257
macroman	MacRoman
macce	MacCentralEurope
utf8	UTF-8
ucs2	UnicodeBig

Warning. Do not issue the query 'set names' with Connector/J, as the driver will not detect that the character set has changed, and will continue to use the character set detected during the initial connection setup.

To allow multiple character sets to be sent from the client, the UTF-8 encoding should be used, either by configuring `utf8` as the default server character set, or by configuring the JDBC driver to use UTF-8 through the *characterEncoding* property.

23.3.4.5. Connecting Securely Using SSL

SSL in MySQL Connector/J encrypts all data (other than the initial handshake) between the JDBC driver and the server. The performance penalty for enabling SSL is an increase in query processing time between 35% and 50%, depending on the size of the query, and the amount of data it returns.

For SSL Support to work, you must have the following:

- A JDK that includes JSSE (Java Secure Sockets Extension), like JDK-1.4.1 or newer. SSL does not currently work with a JDK that you can add JSSE to, like JDK-1.2.x or JDK-1.3.x due to the following JSSE bug: <http://developer.java.sun.com/developer/bugParade/bugs/4273544.html>
- A MySQL server that supports SSL and has been compiled and configured to do so, which is MySQL-4.0.4 or later, see [Section 5.9.7, “Using Secure Connections”](#), for more information.
- A client certificate (covered later in this section)

You will first need to import the MySQL server CA Certificate into a Java truststore. A sample MySQL server CA Certificate is located in the SSL subdirectory of the MySQL source distribution. This is what SSL will use to determine if you are communicating with a secure MySQL server.

To use Java's **keytool** to create a truststore in the current directory , and import the server's CA certificate (cacert.pem), you can do the following (assuming that **keytool** is in your path. The **keytool** should be located in the bin subdirectory of your JDK or JRE):

```
shell> keytool -import -alias mysqlServerCACert -file cacert.pem -ke
```

Keytool will respond with the following information:

```
Enter keystore password:  *****
Owner: EMAILADDRESS=walrus@example.com, CN=Walrus, O=MySQL AB, L=Ore
-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=Walrus, O=MySQL AB, L=Or
e-State, C=RU
Serial number: 0
Valid from: Fri Aug 02 16:55:53 CDT 2002 until: Sat Aug 02 16:55:53
Certificate fingerprints:
    MD5:  61:91:A0:F2:03:07:61:7A:81:38:66:DA:19:C4:8D:AB
    SHA1: 25:77:41:05:D5:AD:99:8C:14:8C:CA:68:9C:2F:B8:89:C3:34
Trust this certificate? [no]: yes
Certificate was added to keystore
```

You will then need to generate a client certificate, so that the MySQL server knows that it is talking to a secure client:

```
shell> keytool -genkey -keyalg rsa -alias mysqlClientCertificate -k
```

Keytool will prompt you for the following information, and create a keystore named keystore in the current directory.

You should respond with information that is appropriate for your situation:

```
Enter keystore password:  *****
What is your first and last name?
[Unknown]: Matthews
What is the name of your organizational unit?
[Unknown]: Software Development
What is the name of your organization?
[Unknown]: MySQL AB
```

```

What is the name of your City or Locality?
[Unknown]: Flossmoor
What is the name of your State or Province?
[Unknown]: IL
What is the two-letter country code for this unit?
[Unknown]: US
Is <CN=Matthews, OU=Software Development, O=MySQL AB,
L=Flossmoor, ST=IL, C=US> correct?
[no]: y

Enter key password for <mysqlClientCertificate>
(RETURN if same as keystore password):

```

Finally, to get JSSE to use the keystore and truststore that you have generated, you need to set the following system properties when you start your JVM, replacing `path_to_keystore_file` with the full path to the keystore file you created, `path_to_truststore_file` with the path to the truststore file you created, and using the appropriate password values for each property.

```

-Djavax.net.ssl.keyStore=path_to_keystore_file
-Djavax.net.ssl.keyStorePassword=*****
-Djavax.net.ssl.trustStore=path_to_truststore_file
-Djavax.net.ssl.trustStorePassword=*****

```

You will also need to set `useSSL` to `true` in your connection parameters for MySQL Connector/J, either by adding `useSSL=true` to your URL, or by setting the property `useSSL` to `true` in the `java.util.Properties` instance you pass to `DriverManager.getConnection()`.

You can test that SSL is working by turning on JSSE debugging (as detailed below), and look for the following key events:

```

...
*** ClientHello, v3.1
RandomCookie: GMT: 1018531834 bytes = { 199, 148, 180, 215, 74, 12
Session ID: {}
Cipher Suites: { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 1
Compression Methods: { 0 }
***

[write] MD5 and SHA1 hashes: len = 59
0000: 01 00 00 37 03 01 3D B6    90 FA C7 94 B4 D7 4A 0C    ...7..=...
0010: 36 F4 00 A8 37 67 D7 40    10 8A E1 BE 84 99 02 D9    6...7g.@..
0020: DB EF CA 13 79 4E 00 00    10 00 05 00 04 00 09 00    ....yN....
0030: 0A 00 12 00 13 00 03 00    11 01 00                    .....
main, WRITE:  SSL v3.1 Handshake, length = 59
main, READ:   SSL v3.1 Handshake, length = 74

```

```

*** ServerHello, v3.1
RandomCookie: GMT: 1018577560 bytes = { 116, 50, 4, 103, 25, 100,
Session ID: {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63
Cipher Suite: { 0, 5 }
Compression Method: 0
***
%% Created: [Session-1, SSL_RSA_WITH_RC4_128_SHA]
** SSL_RSA_WITH_RC4_128_SHA
[read] MD5 and SHA1 hashes: len = 74
0000: 02 00 00 46 03 01 3D B6 43 98 74 32 04 67 19 64 ...F..=.C.
0010: 3A CA 4F B9 B2 64 D7 42 FE 15 53 BB BE 2A AA 03 :.O..d.B..
0020: 84 6E 52 94 A0 5C 20 A3 E3 54 35 51 7F FC FE B2 .nR..\ ..T
0030: B3 44 3F B6 9E 1E 0B 96 4F AA 4C FF 5C 0F E2 18 .D?.....0.
0040: 11 B1 DB 9E B1 BB 8F 00 05 00 .....
main, READ: SSL v3.1 Handshake, length = 1712
...

```

JSSE provides debugging (to STDOUT) when you set the following system property: `-Djavax.net.debug=all` This will tell you what keystores and truststores are being used, as well as what is going on during the SSL handshake and certificate exchange. It will be helpful when trying to determine what is not working when trying to get an SSL connection to happen.

23.3.4.6. Using Master/Slave Replication with ReplicationConnection

Starting with Connector/J 3.1.7, we've made available a variant of the driver that will automatically send queries to a read/write master, or a failover or round-robin loadbalanced set of slaves based on the state of `Connection.getReadOnly()`.

An application signals that it wants a transaction to be read-only by calling `Connection.setReadOnly(true)`, this replication-aware connection will use one of the slave connections, which are load-balanced per-vm using a round-robin scheme (a given connection is sticky to a slave unless that slave is removed from service). If you have a write transaction, or if you have a read that is time-sensitive (remember, replication in MySQL is asynchronous), set the connection to be not read-only, by calling `Connection.setReadOnly(false)` and the driver will ensure that further calls are sent to the master MySQL server. The driver takes care of propagating the current state of autocommit, isolation level, and catalog between all of the connections that it uses to accomplish this load balancing functionality.

To enable this functionality, use the " `com.mysql.jdbc.ReplicationDriver` " class when configuring your application server's connection pool or when creating an instance of a JDBC driver for your standalone application. Because it accepts the same URL format as the standard MySQL JDBC driver, `ReplicationDriver` does not currently work with `java.sql.DriverManager` - based connection creation unless it is the only MySQL JDBC driver registered with the `DriverManager` .

Here is a short, simple example of how `ReplicationDriver` might be used in a standalone application.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.util.Properties;

import com.mysql.jdbc.ReplicationDriver;

public class ReplicationDriverDemo {

    public static void main(String[] args) throws Exception {
        ReplicationDriver driver = new ReplicationDriver();

        Properties props = new Properties();

        // We want this for failover on the slaves
        props.put("autoReconnect", "true");

        // We want to load balance between the slaves
        props.put("roundRobinLoadBalance", "true");

        props.put("user", "foo");
        props.put("password", "bar");

        //
        // Looks like a normal MySQL JDBC url, with a comma-separate
        // of hosts, the first being the 'master', the rest being an
        // of slaves that the driver will load balance against
        //

        Connection conn =
            driver.connect("jdbc:mysql://master,slave1,slave2,slave3
                props);

        //
        // Perform read/write work on the master
        // by setting the read-only flag to "false"
        //
    }
}
```

```

conn.setReadOnly(false);
conn.setAutoCommit(false);
conn.createStatement().executeUpdate("UPDATE some_table ....
conn.commit();

//
// Now, do a query from a slave, the driver automatically pi
// from the list
//

conn.setReadOnly(true);

ResultSet rs = conn.createStatement().executeQuery("SELECT a
.....
}
}

```

23.3.5. Connector/J Notes and Tips

23.3.5.1. Basic JDBC Concepts

This section provides some general JDBC background.

23.3.5.1.1. Connecting to MySQL Using the DriverManager Interface

When you are using JDBC outside of an application server, the `DriverManager` class manages the establishment of Connections.

The `DriverManager` needs to be told which JDBC drivers it should try to make Connections with. The easiest way to do this is to use `Class.forName()` on the class that implements the `java.sql.Driver` interface. With MySQL Connector/J, the name of this class is `com.mysql.jdbc.Driver`. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

The following section of Java code shows how you might register MySQL Connector/J from the `main()` method of your application:

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

```

```

// Notice, do not import com.mysql.jdbc.*
// or you will have problems!

public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations

            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
            // handle the error
        }
    }
}

```

After the driver has been registered with the `DriverManager`, you can obtain a `Connection` instance that is connected to a particular database by calling `DriverManager.getConnection()`:

Example 23.1. Obtaining a connection from the DriverManager

This example shows how you can obtain a `Connection` instance from the `DriverManager`. There are a few different signatures for the `getConnection()` method. You should see the API documentation that comes with your JDK for more specific information on how to use them.

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

... try {
    Connection conn = DriverManager.getConnection("jdbc:mysql:

    // Do something with the Connection

    ....
} catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}

```

Once a `Connection` is established, it can be used to create `Statement` and `PreparedStatement` objects, as well as retrieve metadata about the database.

This is explained in the following sections.

23.3.5.1.2. Using Statements to Execute SQL

Statement objects allow you to execute basic SQL queries and retrieve the results through the `ResultSet` class which is described later.

To create a Statement instance, you call the `createStatement()` method on the `Connection` object you have retrieved via one of the `DriverManager.getConnection()` or `DataSource.getConnection()` methods described earlier.

Once you have a Statement instance, you can execute a `SELECT` query by calling the `executeQuery(String)` method with the SQL you want to use.

To update data in the database, use the `executeUpdate(String SQL)` method. This method returns the number of rows affected by the update statement.

If you don't know ahead of time whether the SQL statement will be a `SELECT` or an `UPDATE/INSERT`, then you can use the `execute(String SQL)` method. This method will return `true` if the SQL query was a `SELECT`, or `false` if it was an `UPDATE`, `INSERT`, or `DELETE` statement. If the statement was a `SELECT` query, you can retrieve the results by calling the `getResultSet()` method. If the statement was an `UPDATE`, `INSERT`, or `DELETE` statement, you can retrieve the affected rows count by calling `getUpdateCount()` on the Statement instance.

Example 23.2. Using `java.sql.Statement` to execute a `SELECT` query

```
// assume that conn is an already created JDBC connection
Statement stmt = null;
ResultSet rs = null;

try {
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");

    // or alternatively, if you don't know ahead of time that
    // the query will be a SELECT...

    if (stmt.execute("SELECT foo FROM bar")) {
        rs = stmt.getResultSet();
    }
}
```

```

    // Now do something with the ResultSet ....
} finally {
    // it is a good idea to release
    // resources in a finally{} block
    // in reverse-order of their creation
    // if they are no-longer needed

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { // ignore }

        rs = null;
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { // ignore }

        stmt = null;
    }
}

```

23.3.5.1.3. Using callableStatements to Execute Stored Procedures

Starting with MySQL server version 5.0 when used with Connector/J 3.1.1 or newer, the `java.sql.CallableStatement` interface is fully implemented with the exception of the `getParameterMetaData()` method.

See [Chapter 17, *Stored Procedures and Functions*](#), for more information on MySQL stored procedures.

Connector/J exposes stored procedure functionality through JDBC's `CallableStatement` interface.

Note. Current versions of MySQL server do not return enough information for the JDBC driver to provide result set metadata for callable statements. This means that when using `CallableStatement`, `ResultSetMetaData` may return `NULL`.

The following example shows a stored procedure that returns the value of `inOutParam` incremented by 1, and the string passed in via `inputParam` as a `ResultSet`:

Example 23.3. Stored Procedures

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), INOUT inOutParam
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
END
```

To use the `demoSp` procedure with Connector/J, follow these steps:

1. Prepare the callable statement by using `Connection.prepareCall()`.

Notice that you have to use JDBC escape syntax, and that the parentheses surrounding the parameter placeholders are not optional:

Example 23.4. Using `Connection.prepareCall()`

```
import java.sql.CallableStatement;

...

//
// Prepare a call to the stored procedure 'demoSp'
// with two parameters
//
// Notice the use of JDBC-escape syntax ({call ...})
//

CallableStatement cStmt = conn.prepareCall("{call demoSp(?,

cStmt.setString(1, "abcdefg");
```

Note. `Connection.prepareCall()` is an expensive method, due to the metadata retrieval that the driver performs to support output parameters. For performance reasons, you should try to minimize unnecessary calls to `Connection.prepareCall()` by reusing `CallableStatement` instances in your code.

2. Register the output parameters (if any exist)

To retrieve the values of output parameters (parameters specified as OUT or INOUT when you created the stored procedure), JDBC requires that they be specified before statement execution using the various `registerOutputParameter()` methods in the `CallableStatement` interface:

Example 23.5. Registering output parameters

```
import java.sql.Types;
...
//
// Connector/J supports both named and indexed
// output parameters. You can register output
// parameters using either method, as well
// as retrieve output parameters using either
// method, regardless of what method was
// used to register them.
//
// The following examples show how to use
// the various methods of registering
// output parameters (you should of course
// use only one registration per parameter).
//
//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//
cStmt.registerOutParameter(2, Types.INTEGER);
//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//
cStmt.registerOutParameter("inOutParam", Types.INTEGER);
...
```

3. Set the input parameters (if any exist)

Input and in/out parameters are set as for `PreparedStatement` objects.

However, CallableStatement also supports setting parameters by name:

Example 23.6. Setting CallableStatement input parameters

```
...  
  
    //  
    // Set a parameter by index  
    //  
    cStmt.setString(1, "abcdefg");  
  
    //  
    // Alternatively, set a parameter using  
    // the parameter name  
    //  
    cStmt.setString("inputParameter", "abcdefg");  
  
    //  
    // Set the 'in/out' parameter using an index  
    //  
    cStmt.setInt(2, 1);  
  
    //  
    // Alternatively, set the 'in/out' parameter  
    // by name  
    //  
    cStmt.setInt("inOutParam", 1);  
  
...
```

4. Execute the CallableStatement, and retrieve any result sets or output parameters.

Although CallableStatement supports calling any of the Statement execute methods (executeUpdate(), executeQuery() or execute()), the most flexible method to call is execute(), as you do not need to know ahead of time if the stored procedure returns result sets:

Example 23.7. Retrieving results and output parameter values

```
...
```

```

boolean hadResults = cStmt.execute();

//
// Process all returned result sets
//

while (hadResults) {
    ResultSet rs = cStmt.getResultSet();

    // process result set
    ...

    hadResults = rs.getMoreResults();
}

//
// Retrieve output parameters
//
// Connector/J supports both index-based and
// name-based retrieval
//

int outputValue = rs.getInt(2); // index-based

outputValue = rs.getInt("inOutParam"); // name-based

...

```

23.3.5.1.4. Retrieving AUTO_INCREMENT Column Values

Before version 3.0 of the JDBC API, there was no standard way of retrieving key values from databases that supported auto increment or identity columns. With older JDBC drivers for MySQL, you could always use a MySQL-specific method on the Statement interface, or issue the query `SELECT LAST_INSERT_ID()` after issuing an `INSERT` to a table that had an `AUTO_INCREMENT` key. Using the MySQL-specific method call isn't portable, and issuing a `SELECT` to get the `AUTO_INCREMENT` key's value requires another round-trip to the database, which isn't as efficient as possible. The following code snippets demonstrate the three different ways to retrieve `AUTO_INCREMENT` values. First, we demonstrate the use of the new JDBC-3.0 method `getGeneratedKeys()` which is now the preferred method to use if you need to retrieve `AUTO_INCREMENT` keys and have access to JDBC-3.0. The second example shows how you can retrieve the same value using a standard `SELECT LAST_INSERT_ID()` query. The final example shows how updatable result sets can retrieve the `AUTO_INCREMENT` value when using the `insertRow()` method.

Example 23.8. Retrieving AUTO_INCREMENT column values using Statement.getGeneratedKeys()

```
Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets assuming you have a
    // Connection 'conn' to a MySQL database already
    // available

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY
                               java.sql.ResultSet.CONCUR_UPDATABLE)

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //

    stmt.executeUpdate(
        "INSERT INTO autoIncTutorial (dataField) "
        + "values ('Can I Get the Auto Increment Field?')",
        Statement.RETURN_GENERATED_KEYS);

    //
    // Example of using Statement.getGeneratedKeys()
    // to retrieve the value of an auto-increment
    // value
    //

    int autoIncKeyFromApi = -1;

    rs = stmt.getGeneratedKeys();

    if (rs.next()) {
        autoIncKeyFromApi = rs.getInt(1);
    }
}
```

```

    } else {
        // throw an exception from here
    }
    rs.close();
    rs = null;
    System.out.println("Key returned from getGeneratedKeys():"
        + autoIncKeyFromApi);
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}

```

Example 23.9. Retrieving AUTO_INCREMENT column values using SELECT LAST_INSERT_ID()

```

Statement stmt = null;
ResultSet rs = null;

try {
    //
    // Create a Statement instance that we can use for
    // 'normal' result sets.

    stmt = conn.createStatement();

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
}

```

```

stmt.executeUpdate(
    "CREATE TABLE autoIncTutorial ("
    + "priKey INT NOT NULL AUTO_INCREMENT, "
    + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

//
// Insert one row that will generate an AUTO INCREMENT
// key in the 'priKey' field
//

stmt.executeUpdate(
    "INSERT INTO autoIncTutorial (dataField) "
    + "values ('Can I Get the Auto Increment Field?')");

//
// Use the MySQL LAST_INSERT_ID()
// function to do the same thing as getGeneratedKeys()
//

int autoIncKeyFromFunc = -1;
rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");

if (rs.next()) {
    autoIncKeyFromFunc = rs.getInt(1);
} else {
    // throw an exception from here
}

rs.close();

System.out.println("Key returned from " + "'SELECT LAST_INSERT_I
    + autoIncKeyFromFunc);

} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}

```

```
}
```

Example 23.10. Retrieving AUTO_INCREMENT column values in Updatable ResultSets

```
Statement stmt = null;
ResultSet rs = null;

try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets as well as an 'updatable'
    // one, assuming you have a Connection 'conn' to
    // a MySQL database already available
    //

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY
                               java.sql.ResultSet.CONCUR_UPDATABLE)

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Example of retrieving an AUTO INCREMENT key
    // from an updatable result set
    //

    rs = stmt.executeQuery("SELECT priKey, dataField "
        + "FROM autoIncTutorial");

    rs.moveToInsertRow();

    rs.updateString("dataField", "AUTO INCREMENT here?");
    rs.insertRow();

    //
    // the driver adds rows at the end
    //

    rs.last();
```

```

//
// We should now be on the row we just inserted
//

int autoIncKeyFromRS = rs.getInt("priKey");

rs.close();

rs = null;

System.out.println("Key returned for inserted row: "
    + autoIncKeyFromRS);
} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}

```

When you run the preceding example code, you should get the following output:
Key returned from getGeneratedKeys(): 1 Key returned from SELECT
LAST_INSERT_ID(): 1 Key returned for inserted row: 2 You should be aware, that
at times, it can be tricky to use the SELECT LAST_INSERT_ID() query, as that
function's value is scoped to a connection. So, if some other query happens on
the same connection, the value will be overwritten. On the other hand, the
getGeneratedKeys() method is scoped by the Statement instance, so it can be
used even if other queries happen on the same connection, but not on the same
Statement instance.

23.3.5.2. Using Connector/J with J2EE and Other Java Frameworks

This section describes how to use Connector/J in several contexts.

23.3.5.2.1. General J2EE Concepts

This section provides general background on J2EE concepts that pertain to use of Connector/J.

23.3.5.2.1.1. Understanding Connection Pooling

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them.

This technique of pooling connections is based on the fact that most applications only need a thread to have access to a JDBC connection when they are actively processing a transaction, which usually take only milliseconds to complete. When not processing a transaction, the connection would otherwise sit idle. Instead, connection pooling allows the idle connection to be used by some other thread to do useful work.

In practice, when a thread needs to do work against a MySQL or other database with JDBC, it requests a connection from the pool. When the thread is finished using the connection, it returns it to the pool, so that it may be used by any other threads that want to use it.

When the connection is loaned out from the pool, it is used exclusively by the thread that requested it. From a programming point of view, it is the same as if your thread called `DriverManager.getConnection()` every time it needed a JDBC connection, however with connection pooling, your thread may end up using either a new, or already-existing connection.

Connection pooling can greatly increase the performance of your Java application, while reducing overall resource usage. The main benefits to connection pooling are:

- Reduced connection creation time

Although this is not usually an issue with the quick connection setup that

MySQL offers compared to other databases, creating new JDBC connections still incurs networking and JDBC driver overhead that will be avoided if connections are recycled.

- Simplified programming model

When using connection pooling, each individual thread can act as though it has created its own JDBC connection, allowing you to use straight-forward JDBC programming techniques.

- Controlled resource usage

If you don't use connection pooling, and instead create a new connection every time a thread needs one, your application's resource usage can be quite wasteful and lead to unpredictable behavior under load.

Remember that each connection to MySQL has overhead (memory, CPU, context switches, and so forth) on both the client and server side. Every connection limits how many resources there are available to your application as well as the MySQL server. Many of these resources will be used whether or not the connection is actually doing any useful work!

Connection pools can be tuned to maximize performance, while keeping resource utilization below the point where your application will start to fail rather than just run slower.

Luckily, Sun has standardized the concept of connection pooling in JDBC through the JDBC-2.0 Optional interfaces, and all major application servers have implementations of these APIs that work fine with MySQL Connector/J.

Generally, you configure a connection pool in your application server configuration files, and access it via the Java Naming and Directory Interface (JNDI). The following code shows how you might use a connection pool from an application deployed in a J2EE application server:

Example 23.11. Using a connection pool with a J2EE application server

```
import java.sql.Connection;  
import java.sql.SQLException;  
import java.sql.Statement;
```

```

import javax.naming.InitialContext;
import javax.sql.DataSource;

public class MyServletJspOrEjb {

    public void doSomething() throws Exception {
        /*
         * Create a JNDI Initial context to be able to
         * lookup the DataSource
         *
         * In production-level code, this should be cached as
         * an instance or static variable, as it can
         * be quite expensive to create a JNDI context.
         *
         * Note: This code only works when you are using servlets
         * or EJBs in a J2EE application server. If you are
         * using connection pooling in standalone Java code, you
         * will have to create/configure datasources using whatever
         * mechanisms your particular connection pooling library
         * provides.
         */

        InitialContext ctx = new InitialContext();

        /*
         * Lookup the DataSource, which will be backed by a pool
         * that the application server provides. DataSource instanc
         * are also a good candidate for caching as an instance
         * variable, as JNDI lookups can be expensive as well.
         */

        DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/M

        /*
         * The following code is what would actually be in your
         * Servlet, JSP or EJB 'service' method...where you need
         * to work with a JDBC connection.
         */

        Connection conn = null;
        Statement stmt = null;

        try {
            conn = ds.getConnection();

            /*
             * Now, use normal JDBC programming to work with
             * MySQL, making sure to close each resource when you're
             * finished with it, which allows the connection pool

```


statements or result sets) are closed, so that they may be re-used, otherwise they will be stranded, which in the best case means that the MySQL server resources they represent (such as buffers, locks, or sockets) may be tied up for some time, or worst case, may be tied up forever.

What's the Best Size for my Connection Pool?

As with all other configuration rules-of-thumb, the answer is: it depends. Although the optimal size depends on anticipated load and average database transaction time, the optimum connection pool size is smaller than you might expect. If you take Sun's Java Petstore blueprint application for example, a connection pool of 15-20 connections can serve a relatively moderate load (600 concurrent users) using MySQL and Tomcat with response times that are acceptable.

To correctly size a connection pool for your application, you should create load test scripts with tools such as Apache JMeter or The Grinder, and load test your application.

An easy way to determine a starting point is to configure your connection pool's maximum number of connections to be unbounded, run a load test, and measure the largest amount of concurrently used connections. You can then work backward from there to determine what values of minimum and maximum pooled connections give the best performance for your particular application.

23.3.5.2.2. Using Connector/J with Tomcat

The following instructions are based on the instructions for Tomcat-5.x, available at <http://jakarta.apache.org/tomcat/tomcat-5.0-doc/jndi-datasource-examples-howto.html> which is current at the time this document was written.

First, install the .jar file that comes with Connector/J in `$CATALINA_HOME/common/lib` so that it is available to all applications installed in the container.

Next, Configure the JNDI DataSource by adding a declaration resource to `$CATALINA_HOME/conf/server.xml` in the context that defines your web application:

```
<Context . . . .>
```

...

```
<Resource name="jdbc/MySQLDB"
          auth="Container"
          type="javax.sql.DataSource"/>
```

```
<!-- The name you used above, must match _exactly_ here!
```

```
    The connection pool will be bound into JNDI with the name
    "java:/comp/env/jdbc/MySQLDB"
-->
```

```
<ResourceParams name="jdbc/MySQLDB">
  <parameter>
    <name>factory</name>
    <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
  </parameter>
```

```
<!-- Don't set this any higher than max_connections on your
MySQL server, usually this should be a 10 or a few 10's
of connections, not hundreds or thousands -->
```

```
<parameter>
  <name>maxActive</name>
  <value>10</value>
</parameter>
```

```
<!-- You don't want to many idle connections hanging around
if you can avoid it, only enough to soak up a spike in
the load -->
```

```
<parameter>
  <name>maxIdle</name>
  <value>5</value>
</parameter>
```

```
<!-- Don't use autoReconnect=true, it's going away eventually
and it's a crutch for older connection pools that couldn't
test connections. You need to decide whether your applicati
supposed to deal with SQLExceptions (hint, it should), and
how much of a performance penalty you're willing to pay
to ensure 'freshness' of the connection -->
```

```
<parameter>
  <name>validationQuery</name>
  <value>SELECT 1</value>
</parameter>
```

```
<!-- The most conservative approach is to test connections
before they're given to your application. For most applicati
this is okay, the query used above is very small and takes
no real server resources to process, other than the time use
to traverse the network.
```

```
    If you have a high-load application you'll need to rely on
something else. -->
```

```
<parameter>
  <name>testOnBorrow</name>
  <value>>true</value>
</parameter>
```

```
<!-- Otherwise, or in addition to testOnBorrow, you can test
while connections are sitting idle -->
```

```
<parameter>
  <name>testWhileIdle</name>
  <value>>true</value>
</parameter>
```

```
<!-- You have to set this value, otherwise even though
you've asked connections to be tested while idle,
the idle evicter thread will never run -->
```

```
<parameter>
  <name>timeBetweenEvictionRunsMillis</name>
  <value>10000</value>
</parameter>
```

```
<!-- Don't allow connections to hang out idle too long,
never longer than what wait_timeout is set to on the
server...A few minutes or even fraction of a minute
is sometimes okay here, it depends on your application
and how much spikey load it will see -->
```

```
<parameter>
  <name>minEvictableIdleTimeMillis</name>
  <value>60000</value>
</parameter>
```

```
<!-- Username and password used when connecting to MySQL -->
```

```
<parameter>
  <name>username</name>
  <value>someuser</value>
</parameter>
```

```

<parameter>
  <name>password</name>
  <value>somepass</value>
</parameter>

<!-- Class name for the Connector/J driver -->

<parameter>
  <name>driverClassName</name>
  <value>com.mysql.jdbc.Driver</value>
</parameter>

<!-- The JDBC connection url for connecting to MySQL, notice
that if you want to pass any other MySQL-specific parameter
you should pass them here in the URL, setting them using th
parameter tags above will have no effect, you will also
need to use &amp; to separate parameter values as the
ampersand is a reserved character in XML -->

<parameter>
  <name>url</name>
  <value>jdbc:mysql://localhost:3306/test</value>
</parameter>

</ResourceParams>
</Context>

```

In general, you should follow the installation instructions that come with your version of Tomcat, as the way you configure datasources in Tomcat changes from time-to-time, and unfortunately if you use the wrong syntax in your XML file, you will most likely end up with an exception similar to the following:

```

Error: java.sql.SQLException: Cannot load JDBC driver class 'null '
state: null

```

23.3.5.2.3. Using Connector/J with JBoss

These instructions cover JBoss-4.x. To make the JDBC driver classes available to the application server, copy the .jar file that comes with Connector/J to the `lib` directory for your server configuration (which is usually called `default`). Then, in the same configuration directory, in the subdirectory named `deploy`, create a datasource configuration file that ends with `-ds.xml`, which tells JBoss to deploy this file as a JDBC Datasource. The file should have the following contents:

```

<datasources>
  <local-tx-datasource>
    <!-- This connection pool will be bound into JNDI with the n
         "java:/MySQLDB" -->

    <jndi-name>MySQLDB</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/dbname</connecti
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>user</user-name>
    <password>pass</password>

    <min-pool-size>5</min-pool-size>

    <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->

    <max-pool-size>20</max-pool-size>

    <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes is usually okay here,
         it depends on your application
         and how much spikey load it will see -->

    <idle-timeout-minutes>5</idle-timeout-minutes>

    <!-- If you're using Connector/J 3.1.8 or newer, you can use
         our implementation of these to increase the robustness
         of the connection pool. -->

    <exception-sorter-class-name>com.mysql.jdbc.integration.jbos
    <valid-connection-checker-class-name>com.mysql.jdbc.integrat

  </local-tx-datasource>
</datasources>

```

23.3.5.3. Common Problems and Solutions

There are a few issues that seem to be commonly encountered often by users of MySQL Connector/J. This section deals with their symptoms, and their resolutions.

24.3.5.3.1:

Question:

When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What's going on? I can connect just fine with the MySQL command-line client.

Answer:

MySQL Connector/J must use TCP/IP sockets to connect to MySQL, as Java does not support Unix Domain Sockets. Therefore, when MySQL Connector/J connects to MySQL, the security manager in MySQL server will use its grant tables to determine whether the connection should be allowed.

You must add the necessary security credentials to the MySQL server for this to happen, using the GRANT statement to your MySQL Server. See [Section 13.5.1.3, “GRANT Syntax”](#), for more information.

Note. Testing your connectivity with the **mysql** command-line client will not work unless you add the `--host` flag, and use something other than `localhost` for the host. The **mysql** command-line client will use Unix domain sockets if you use the special hostname `localhost`. If you are testing connectivity to `localhost`, use `127.0.0.1` as the hostname instead.

Warning. Changing privileges and permissions improperly in MySQL can potentially cause your server installation to not have optimal security properties.

24.3.5.3.2:

Question:

My application throws an SQLException 'No Suitable Driver'. Why is this happening?

Answer:

There are three possible causes for this error:

- The Connector/J driver is not in your CLASSPATH, see [Section 23.3.2](#),

[“Installing Connector/J”](#).

- The format of your connection URL is incorrect, or you are referencing the wrong JDBC driver.
- When using DriverManager, the `jdbc.drivers` system property has not been populated with the location of the Connector/J driver.

24.3.5.3.3:

Question:

I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.  
Is there a MySQL server running on the machine/port you  
are trying to connect to?
```

```
(java.security.AccessControlException)  
SQLState: 08S01  
VendorError: 0
```

Answer:

Either you're running an Applet, your MySQL server has been installed with the "--skip-networking" option set, or your MySQL server has a firewall sitting in front of it.

Applets can only make network connections back to the machine that runs the web server that served the .class files for the applet. This means that MySQL must run on the same machine (or you must have some sort of port re-direction) for this to work. This also means that you will not be able to test applets from your local file system, you must always deploy them to a web server.

MySQL Connector/J can only communicate with MySQL using TCP/IP, as Java does not support Unix domain sockets. TCP/IP communication with MySQL might be affected if MySQL was started with the "--skip-networking" flag, or if it is firewalled.

If MySQL has been started with the "--skip-networking" option set (the Debian Linux package of MySQL server does this for example), you need to comment it

out in the file `/etc/mysql/my.cnf` or `/etc/my.cnf`. Of course your `my.cnf` file might also exist in the data directory of your MySQL server, or anywhere else (depending on how MySQL was compiled for your system). Binaries created by MySQL AB always look in `/etc/my.cnf` and `[datadir]/my.cnf`. If your MySQL server has been firewalled, you will need to have the firewall configured to allow TCP/IP connections from the host where your Java code is running to the MySQL server on the port that MySQL is listening to (by default, 3306).

24.3.5.3.4:

Question:

I have a servlet/application that works fine for a day, and then stops working overnight

Answer:

MySQL closes connections after 8 hours of inactivity. You either need to use a connection pool that handles stale connections or use the "autoReconnect" parameter (see [Section 23.3.4.1, "Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J"](#)).

Also, you should be catching `SQLException`s in your application and dealing with them, rather than propagating them all the way until your application exits, this is just good programming practice. MySQL Connector/J will set the `SQLState` (see `java.sql.SQLException.getSQLState()` in your APIDocs) to "08S01" when it encounters network-connectivity issues during the processing of a query. Your application code should then attempt to re-connect to MySQL at this point.

The following (simplistic) example shows what code that can handle these exceptions might look like:

Example 23.12. Example of transaction with retry logic

```
public void doBusinessOp() throws SQLException {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;

    //
```

```

// How many times do you want to retry the transaction
// (or at least _getting_ a connection)?
//
int retryCount = 5;

boolean transactionCompleted = false;

do {
    try {
        conn = getConnection(); // assume getting this from
                                // javax.sql.DataSource, or
                                // java.sql.DriverManager

        conn.setAutoCommit(false);

        //
        // Okay, at this point, the 'retry-ability' of the
        // transaction really depends on your application logic
        // whether or not you're using autocommit (in this case
        // not), and whether you're using transactional storage
        // engines
        //
        // For this example, we'll assume that it's _not_ safe
        // to retry the entire transaction, so we set retry
        // to 0 at this point
        //
        // If you were using exclusively transaction-safe tables
        // or your application could recover from a connection
        // bad in the middle of an operation, then you would
        // touch 'retryCount' here, and just let the loop retry
        // until retryCount == 0.
        //
        retryCount = 0;

        stmt = conn.createStatement();

        String query = "SELECT foo FROM bar ORDER BY baz";

        rs = stmt.executeQuery(query);

        while (rs.next()) {
        }

        rs.close();
        rs = null;

        stmt.close();
        stmt = null;

        conn.commit();
    }
}

```

```

        conn.close();
        conn = null;

        transactionCompleted = true;
    } catch (SQLException sqlEx) {

        //
        // The two SQL states that are 'retry-able' are 08S0
        // for a communications error, and 40001 for deadloc
        //
        // Only retry if the error was due to a stale connec
        // communications problem or deadlock
        //

        String sqlState = sqlEx.getSQLState();

        if ("08S01".equals(sqlState) || "40001".equals(sqlSt
            retryCount--;
        } else {
            retryCount = 0;
        }
    } finally {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException sqlEx) {
                // You'd probably want to log this . . .
            }
        }

        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException sqlEx) {
                // You'd probably want to log this as well .
            }
        }

        if (conn != null) {
            try {
                //
                // If we got here, and conn is not null, the
                // transaction should be rolled back, as not
                // all work has been done

                try {
                    conn.rollback();
                } finally {
                    conn.close();
                }
            }
        }
    }
}

```

```

        } catch (SQLException sqlEx) {
            //
            // If we got an exception here, something
            // pretty serious is going on, so we better
            // pass it up the stack, rather than just
            // logging it. . .

            throw sqlEx;
        }
    }
} while (!transactionCompleted && (retryCount > 0));
}

```

Note. Use of the `autoReconnect` option is not recommended because there is no safe method of reconnecting to the MySQL server without risking some corruption of the connection state or database state information. Instead, you should use a connection pool which will enable your application to connect to the MySQL server using an available connection from the pool. The `autoReconnect` facility is deprecated, and may be removed in a future release.

24.3.5.3.5:

Question:

I'm trying to use JDBC-2.0 updatable result sets, and I get an exception saying my result set is not updatable.

Answer:

Because MySQL does not have row identifiers, MySQL Connector/J can only update result sets that have come from queries on tables that have at least one primary key, the query must select every primary key and the query can only span one table (that is, no joins). This is outlined in the JDBC specification.

23.3.6. Connector/J Support

23.3.6.1. Connector/J Community Support

MySQL AB provides assistance to the user community by means of its mailing lists. For Connector/J related issues, you can get help from experienced users by using the MySQL and Java mailing list. Archives and subscription information is

available online at <http://lists.mysql.com/java>.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [Section 1.7.1, “MySQL Mailing Lists”](#).

Community support from experienced users is also available through the [JDBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [Section 1.7.2, “MySQL Community Support at the MySQL Forums”](#).

23.3.6.2. How to Report Connector/J Bugs or Problems

The normal place to report bugs is <http://bugs.mysql.com/>, which is the address for our bugs database. This database is public, and can be browsed and searched by anyone. If you log in to the system, you will also be able to enter new reports.

If you have found a sensitive security bug in MySQL, you can send email to security_at_mysql.com.

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release.

This section will help you write your report correctly so that you don't waste your time doing things that may not help us much or at all.

If you have a repeatable bug report, please report it to the bugs database at <http://bugs.mysql.com/>. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

To report other problems, you can use one of the MySQL mailing lists.

Remember that it is possible for us to respond to a message containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details don't matter.

A good principle is this: If you are in doubt about stating something, state it. It is

faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of Connector/J or MySQL used, and (b) not fully describing the platform on which Connector/J is installed (including the JVM version, and the platform type and version number that MySQL itself is installed on).

This is highly relevant information, and in 99 cases out of 100, the bug report is useless without it. Very often we get questions like, “Why doesn't this work for me?” Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has already been fixed in newer MySQL versions.

Sometimes the error is platform-dependent; in such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If at all possible, you should create a repeatable, standalone testcase that doesn't involve any third-party classes.

To streamline this process, we ship a base class for testcases with Connector/J, named `'com.mysql.jdbc.util.BaseBugReport'`. To create a testcase for Connector/J using this class, create your own class that inherits from `com.mysql.jdbc.util.BaseBugReport` and override the methods `setUp()`, `tearDown()` and `runTest()`.

In the `setUp()` method, create code that creates your tables, and populates them with any data needed to demonstrate the bug.

In the `runTest()` method, create code that demonstrates the bug using the tables and data you created in the `setUp` method.

In the `tearDown()` method, drop any tables you created in the `setUp()` method.

In any of the above three methods, you should use one of the variants of the `getConnection()` method to create a JDBC connection to MySQL:

- `getConnection()` - Provides a connection to the JDBC URL specified in

`getUrl()`. If a connection already exists, that connection is returned, otherwise a new connection is created.

- `getConnection()` - Use this if you need to get a new connection for your bug report (i.e. there's more than one connection involved).
- `getConnection(String url)` - Returns a connection using the given URL.
- `getConnection(String url, Properties props)` - Returns a connection using the given URL and properties.

If you need to use a JDBC URL that is different from `'jdbc:mysql:///test'`, override the method `getUrl()` as well.

Use the `assertTrue(boolean expression)` and `assertTrue(String failureMessage, boolean expression)` methods to create conditions that must be met in your testcase demonstrating the behavior you are expecting (vs. the behavior you are observing, which is why you are most likely filing a bug report).

Finally, create a `main()` method that creates a new instance of your testcase, and calls the `run` method:

```
public static void main(String[] args) throws Exception {
    new MyBugReport().run();
}
```

Once you have finished your testcase, and have verified that it demonstrates the bug you are reporting, upload it with your bug report to <http://bugs.mysql.com/>.

23.3.6.3. Connector/J Change History

The Connector/J Change History (Changelog) is located with the main Changelog for MySQL. See [Section D.5, “MySQL Connector/J Change History”](#).

23.4. MySQL Connector/MXJ

MySQL Connector/MXJ is a solution for deploying the MySQL database engine (mysqld) intelligently from within a Java package.

23.4.1. Introduction to Connector/MXJ

MySQL Connector/MXJ is a Java Utility package for deploying and managing a MySQL database. Deploying and using MySQL can be as easy as adding an additional parameter to the JDBC connection url, which will result in the database being started when the first connection is made. This makes it easy for Java developers to deploy applications which require a database by reducing installation barriers for their end-users.

MySQL Connector/MXJ makes the MySQL database appear to be a java-based component. It does this by determining what platform the system is running on, selecting the appropriate binary, and launching the executable. It will also optionally deploy an initial database, with any specified parameters.

Included are instructions for use with a JDBC driver and deploying as a JMX MBean to JBoss.

You can download sources and binaries from:
<http://dev.mysql.com/downloads/connector/mxj/>

This a beta release and feedback is welcome and encouraged.

Please send questions or comments to the [MySQL and Java mailing list](#).

23.4.1.1. Connector/MXJ Versions

Connector/MX

- Connector/MXJ 5.x, currently in beta status, includes mysqld version 5.0.22 and includes binaries for Linux x86, Mac OS X PPC, Windows XP/NT/2000 x86 and Solaris SPARC. Connector/MXJ 5.x requires the Connector/J 5.x package.

- Connector/MXJ 1.x includes `mysqld` version 4.1.13 and includes binaries for Linux x86, Windows XP/NT/2000 x86 and Solaris SPARC. Connector/MXJ 1.x requires the Connector/J 3.x package.

This guide provides information on the Connector/MXJ 5.x release. For information on using the older releases, please see the documentation included with the appropriate distribution.

23.4.1.2. Connector/MXJ Overview

Connector/MXJ consists of a Java class, a copy of the `mysqld` binary for a specific list of platforms, and associated files and support utilities. The Java class controls the initialization of an instance of the embedded `mysqld` binary, and the ongoing management of the `mysqld` process. The entire sequence and management can be controlled entirely from within Java using the Connector/MXJ Java classes. You can see an overview of the contents of the Connector/MXJ package in the figure below.



It is important to note that Connector/MXJ is not an embedded version of MySQL, or a version of MySQL written as part of a Java class. Connector/MXJ works through the use of an embedded, compiled binary of `mysqld` as would normally be used when deploying a standard MySQL installation.

It is the Connector/MXJ wrapper, support classes and tools, that enable Connector/MXJ to appear as a MySQL instance.

When Connector/MXJ is initialized, the corresponding `mysqld` binary for the current platform is extracted, along with a pre-configured data directory. Both are contained within the Connector/MXJ JAR file. The `mysqld` instance is then started, with any additional options as specified during the initialization, and the MySQL database becomes accessible.

Because Connector/MXJ works in combination with Connector/J, you can access and integrate with the MySQL instance through a JDBC connection. When you have finished with the server, the instance is terminated, and, by default, any data created during the session is retained within the temporary directory created when the instance was started.

Connector/MXJ and the embedded `mysqld` instance can be deployed in a number of environments where relying on an existing database, or installing a MySQL instance would be impossible, including CD-ROM embedded database applications and temporary database requirements within a Java-based application environment.

23.4.2. Installing Connector/MXJ

Connector/MXJ does not have a installation application or process, but there are some steps you can follow to make the installation and deployment of Connector/MXJ easier.

Before you start, there are some baseline requirements for

- Java Runtime Environment (v1.4.0 or newer) if you are only going to deploy the package.
- Java Development Kit (v1.4.0 or newer) if you want to build Connector/MXJ from source.
- Connector/J 5.0 or newer.

Depending on your target installation/deployment environment you may also require:

- JBoss - 4.0rc1 or newer
- Apache Tomcat - 5.0 or newer
- Sun's JMX reference implementation version 1.2.1 (from <http://java.sun.com/products/JavaManagement/>)

23.4.2.1. Supported Platforms

Connector/MXJ is compatible with any platform supporting Java and MySQL. By default, Connector/MXJ incorporates the `mysqld` binary for a select number of platforms, as outlined below.

- Linux, i386

- Windows NT, Windows 2000, Windows XP, x86
- Solaris 8, SPARC 32 (compatible with Solaris 8, Solaris 9 and Solaris 10 on SPARC 32-bit and 64-bit platforms)
- Mac OS X, PowerPC

For more information on packaging your own Connector/MXJ with the platforms you require, see [Section 23.4.5.1, “Creating your own Connector/MXJ Package”](#)

23.4.2.2. Connector/MXJ Base Installation

Because there is no formal installation process, the method, installation directory, and access methods you use for Connector/MXJ are entirely up to your individual requirements.

To perform a basic installation, choose a target directory for the files included in the Connector/MXJ package. On Unix/Linux systems you may opt to use a directory such as `/usr/local/connector-mxj`; On Windows, you may want to install the files in the base directory, `C:\Connector-MXJ`, or within the Program Files directory.

To install the files:

1. Download the Connector/MXJ package, either in Tar/Gzip format (ideal for Unix/Linux systems) or Zip format (Windows).
2. Extract the files from the package. This will create a directory `connector-mxj`. Copy and optionally rename this directory to your desired location.
3. For best results, you should update your global `CLASSPATH` variable with the location of the Connector/MXJ JAR (`connector-mxj.jar`). You will also need to add the AspectJ Runtime, located in `lib/aspectjrt.jar`.

Within Unix/Linux you can do this globally by editing the global shell profile, or on a user by user basis by editing their individual shell profile.

On Windows 2000, Windows NT and Windows XP, you can edit the global `CLASSPATH` by editing the Environment variables configured through the

System control panel.

23.4.2.3. Connector/MXJ Quick Start Guide

Once you have extracted the Connector/MXJ and Connector/J components you can run one of the sample applications that initiates a MySQL instance. You can test the installation by running the ConnectorMXJUrlTestExample:

```
java ConnectorMXJUrlTestExample
jdbc:mysql:mxj://localhost:3336/test?serverbasedir=/var/tmp/test-mx
[MysqldResource] launching mysqld (getOptions)
[/var/tmp/test-mxj/bin/mysqld][--no-defaults][--pid-file=/var/tmp/te
[MysqldResource] launching mysqld (driver_launched_mysqld_1)
InnoDB: The first specified data file ./ibdata1 did not exist:
InnoDB: a new database to be created!
060726 15:40:42 InnoDB: Setting file ./ibdata1 size to 10 MB
InnoDB: Database physically writes the file full: wait...
060726 15:40:43 InnoDB: Log file ./ib_logfile0 did not exist: new t
InnoDB: Setting log file ./ib_logfile0 size to 5 MB
InnoDB: Database physically writes the file full: wait...
060726 15:40:43 InnoDB: Log file ./ib_logfile1 did not exist: new t
InnoDB: Setting log file ./ib_logfile1 size to 5 MB
InnoDB: Database physically writes the file full: wait...
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
060726 15:40:44 InnoDB: Started; log sequence number 0 0
060726 15:40:44 [Note] /var/tmp/test-mxj/bin/mysqld: ready for conne
Version: '5.0.22-max' socket: 'mysql.sock' port: 3336 MySQL Commu
[MysqldResource] mysqld running as process: 1210
```

5.0.22-max

```
-----
[MysqldResource] stopping mysqld (process: 1210)
060726 15:40:44 [Note] /var/tmp/test-mxj/bin/mysqld: Normal shutdown

060726 15:40:45 InnoDB: Starting shutdown...
060726 15:40:48 InnoDB: Shutdown completed; log sequence number 0 4
060726 15:40:48 [Note] /var/tmp/test-mxj/bin/mysqld: Shutdown comple

[MysqldResource] clearing options
[MysqldResource] shutdown complete
```

The above output shows an instance of MySQL starting, the necessary files

being created (log files, InnoDB data files) and the MySQL database entering the running state. The instance is then shutdown by Connector/MXJ before the example terminates.

23.4.2.4. Deploying Connector/MXJ using Driver Launch

Connector/MXJ and Connector/J work together to enable you to launch an instance of the `mysqld` server through the use of a keyword in the JDBC connection string. Deploying Connector/MXJ within a Java application can be automated through this method, making the deployment of Connector/MXJ a simple process:

1. Download and unzip Connector/MXJ, add `connector-mxj.jar` to the CLASSPATH.
2. To the JDBC connection string, embed the `mxj` keyword, for example:
`jdbc:mysql:mxj://localhost:PORT/DBNAME.`

For more details, see [Section 23.4.3, “Connector/MXJ Configuration”](#).

23.4.2.5. Deploying Connector/MXJ within JBoss

For deployment within a JBoss environment, you must configure the JBoss environment to use the Connector/MXJ component within the JDBC parameters:

1. Download Connector/MXJ copy the `connector-mxj.jar` file to the `$JBOSS_HOME/server/default/lib` directory.
2. Download Connector/J copy the `connector-mxj.jar` file to the `$JBOSS_HOME/server/default/lib` directory.
3. Create an MBean service xml file in the `$JBOSS_HOME/server/default/deploy` directory with any attributes set, for instance the `datadir` and `autostart`.
4. Set the JDBC parameters of your web application to use:

```
String driver = "com.mysql.jdbc.Driver";
String url = "jdbc:mysql:///test?propertiesTransform="+
            "com.mysql.management.jmx.ConnectorMXJPropertiesTra
```

```
String user = "root";
String password = "";
Class.forName(driver);
Connection conn = DriverManager.getConnection(url, user, password);
```

You may wish to create a separate users and database table spaces for each application, rather than using "root and test".

We highly suggest having a routine backup procedure for backing up the database files in the `datadir`.

23.4.2.6. Verifying Installation using JUnit

The best way to ensure that your platform is supported is to run the JUnit tests. These will test the Connector/MXJ classes and the associated components.

23.4.2.6.1. JUnit Test Requirements

The first thing to do is make sure that the components will work on the platform. The `MySQLResource` class is really a wrapper for a native version of MySQL, so not all platforms are supported. At the time of this writing, Linux on the i386 architecture has been tested and seems to work quite well, as does OS X v10.3. There has been limited testing on Windows and Solaris.

Requirements:

1. JDK-1.4 or newer (or the JRE if you aren't going to be compiling the source or JSPs).
2. MySQL Connector/J version 5.0 or newer (from <http://dev.mysql.com/downloads/connector/j/>) installed and available via your CLASSPATH.
3. The `javax.management` classes for JMX version 1.2.1, these are present in the following application servers:
 - JBoss - 4.0rc1 or newer.
 - Apache Tomcat - 5.0 or newer.

- Sun's JMX reference implementation version 1.2.1 (from <http://java.sun.com/products/JavaManagement/>).

4. JUnit 3.8.1 (from <http://www.junit.org/>).

If building from source, All of the requirements from above, plus:

1. Ant version 1.5 or newer (download from <http://ant.apache.org/>).

23.4.2.6.2. Running the JUnit Tests

1. The tests attempt to launch MySQL on the port 3336. If you have a MySQL running, it may conflict, but this isn't very likely because the default port for MySQL is 3306. However, You may set the "c-mxj_test_port" Java property to a port of your choosing. Alternatively, you may wish to start by shutting down any instances of MySQL you have running on the target machine.

The tests suppress output to the console by default. For verbose output, you may set the "c-mxj_test_silent" Java property to "false".

2. To run the JUnit test suite, the \$CLASSPATH must include the following:
 - JUnit
 - JMX
 - Connector/J
 - MySQL Connector/MXJ
3. If connector-mxj.jar is not present in your download, unzip MySQL Connector/MXJ source archive.

```
cd mysqljmx
ant dist
```

Then add \$TEMP/cmjx/stage/connector-mxj/connector-mxj.jar to the CLASSPATH.

4. if you have junit, execute the unit tests. From the command line, type:

```
java junit.textui.TestRunner com.mysql.management.AllTestsSuite
```

The output should look something like this:

```
.....  
.....  
.....  
Time: 259.438  
  
OK (101 tests)
```

Note that the tests are a bit slow near the end, so please be patient.

23.4.3. Connector/MXJ Configuration

23.4.3.1. Running as part of the JDBC Driver

A feature of the MySQL Connector/J JDBC driver is the ability to specify a connection to an embedded Connector/MXJ instance through the use of the `mxj` keyword in the JDBC connection string.

In the following example, we have a program which creates a connection, executes a query, and prints the result to the `System.out`. The MySQL database will be deployed and started as part of the connection process, and shutdown as part of the finally block.

You can find this file in the Connector/MXJ package as `src/ConnectorMXJUrlTestExample.java`.

```
import java.io.File;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.Statement;  
  
import com.mysql.management.driverlaunched.ServerLauncherSocketFacto  
  
public class ConnectorMXJUrlTestExample {  
    public static String DRIVER = "com.mysql.jdbc.Driver";
```

```

public static String JAVA_IO_TMPDIR = "java.io.tmpdir";

public static void main(String[] args) throws Exception {
    File ourAppDir = new File(System.getProperty(JAVA_IO_TMPDIR))
    File databaseDir = new File(ourAppDir, "test-mxj");
    int port = 3336;

    String url = "jdbc:mysql:mxj://localhost:" + port + "/test"
        + "serverbasedir=" + databaseDir;

    System.out.println(url);

    String userName = "root";
    String password = "";

    Class.forName(DRIVER);
    Connection conn = null;
    try {
        conn = DriverManager.getConnection(url, userName, password);
        printQueryResults(conn, "SELECT VERSION()");
    } finally {
        try {
            if (conn != null)
                conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        ServerLauncherSocketFactory.shutdown(databaseDir, null);
    }
}

public static void printQueryResults(Connection conn, String SQL
    throws Exception {
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(SQLquery);
    int columns = rs.getMetaData().getColumnCount();
    System.out.println("-----");
    System.out.println();
    while (rs.next()) {
        for (int i = 1; i <= columns; i++) {
            System.out.println(rs.getString(i));
        }
        System.out.println();
    }
    rs.close();
    stmt.close();
    System.out.println("-----");
    System.out.flush();
}

```

```
        Thread.sleep(100); // wait for System.out to finish flush
    }
}
```

To run the above program, be sure to have connector-mxj.jar and Connector/J in the CLASSPATH. Then type:

```
java ConnectorMXJTestExample
```

23.4.3.2. Running within a Java Object

If you have a java application and wish to “embed” a MySQL database, make use of the `com.mysql.management.MysqldResource` class directly. This class may be instantiated with the default (no argument) constructor, or by passing in a `java.io.File` object representing the directory you wish the server to be “unzipped” into. It may also be instantiated with printstreams for “stdout” and “stderr” for logging.

Once instantiated, a `java.util.Map`, the object will be able to provide a `java.util.Map` of server options appropriate for the platform and version of MySQL which you will be using.

The `MysqldResource` enables you to “start” MySQL with a `java.util.Map` of server options which you provide, as well as “shutdown” the database. The following example shows a simplistic way to embed MySQL in an application using plain java objects.

You can find this file in the Connector/MXJ package as `src/ConnectorMXJObjectTestExample.java`.

```
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Map;

import com.mysql.management.MysqldResource;

public class ConnectorMXJObjectTestExample {
    public static String DRIVER = "com.mysql.jdbc.Driver";
```

```

public static String JAVA_IO_TMPDIR = "java.io.tmpdir";

public static void main(String[] args) throws Exception {
    File ourAppDir = new File(System.getProperty(JAVA_IO_TMPDIR)
    File databaseDir = new File(ourAppDir, "mysql-mxj");
    int port = 3336;

    MysqlDResource mysqlDResource = startDatabase(databaseDir, p

    String userName = "root";
    String password = "";

    Class.forName(DRIVER);
    Connection conn = null;
    try {
        String url = "jdbc:mysql://localhost:" + port + "/test";
        conn = DriverManager.getConnection(url, userName, passwo
        printQueryResults(conn, "SELECT VERSION()");
    } finally {
        try {
            if (conn != null) {
                conn.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            mysqlDResource.shutdown();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

public static MysqlDResource startDatabase(File databaseDir, int
    MysqlDResource mysqlDResource = new MysqlDResource(databaseD

    Map database_options = new HashMap();
    database_options.put("port", Integer.toString(port));
    mysqlDResource.start("test-mysql-thread", database_options)

    if (!mysqlDResource.isRunning()) {
        throw new RuntimeException("MySQL did not start.");
    }

    System.out.println("MySQL is running.");

    return mysqlDResource;
}

```

```

}

public static void printQueryResults(Connection conn, String SQL
    throws Exception {
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(SQLquery);
    int columns = rs.getMetaData().getColumnCount();
    System.out.println("-----");
    System.out.println();
    while (rs.next()) {
        for (int i = 1; i <= columns; i++) {
            System.out.println(rs.getString(i));
        }
        System.out.println();
    }
    rs.close();
    stmt.close();
    System.out.println("-----");
    System.out.flush();
    Thread.sleep(100); // wait for System.out to finish flush
}
}

```

23.4.3.3. Setting server options

Of course there are many options we may wish to set for a MySQL database. These options may be specified as part of the JDBC connection string simply by prefixing each server option with "server.". In the following example we set two driver parameters and two server parameters:

```

String url = "jdbc:mysql://" + hostColonPort + "/"
    + "?"
    + "cacheServerConfiguration=true"
    + "&"
    + "useLocalSessionState=true"
    + "&"
    + "server.basedir=/opt/myapp/db"
    + "&"
    + "server.datadir=/mnt/bigdisk/myapp/data";

```

23.4.4. Connector/MXJ Reference

23.4.4.1. MysqldResource API

23.4.4.1.1. MysqldResource Constructors

The MysqldResource class supports three different constructor forms:

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString, PrintStream out, PrintStream err, Utils util)`

The most detailed constructor, enables you to set the base directory, data directory, select a server by its version string, standard out and standard error and MySQL utilities class.

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString, PrintStream out, PrintStream err)`

Enables you to set the base directory, data directory, select a server by its version string, standard out and standard error.

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString)`

Enables you to set the base directory, data directory and select a server by its version string. Output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource(File baseDir, File dataDir)`

Enables you to set the base directory and data directory. The default MySQL version is selected, and output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource(File baseDir);`

Allows the setting of the "basedir" to deploy the MySQL files to. Output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource();`

The basedir is defaulted to a subdirectory of the java.io.tmpdir. Output for standard out and standard err are directed to System.out and System.err;

23.4.4.1.2. MysqlResource Methods

MysqlResource API includes the following methods:

- `void start(String threadName, Map mysqlArgs);`

Deploys and starts MySQL. The "threadName" string is used to name the thread which actually performs the execution of the MySQL command line. The map is the set of arguments and their values to be passed to the command line.

- `void shutdown();`

Shuts down the MySQL instance managed by the MysqlResource object.

- `Map getServerOptions();`

Returns a map of all the options and their current (or default, if not running) options available for the MySQL database.

- `boolean isRunning();`

Returns true if the MySQL database is running.

- `boolean isReadyForConnections();`

Returns true once the database reports that is ready for connections.

- `void setKillDelay(int millis);`

The default "Kill Delay" is 30 seconds. This represents the amount of time to wait between the initial request to shutdown and issuing a "force kill" if the database has not shutdown by itself.

- `void addCompletionListener(Runnable listener);`

Allows for applications to be notified when the server process completes. Each "listener" will be fired off in its own thread.

- `String getVersion();`

Returns the version of MySQL.

- `void setVersion(int MajorVersion, int minorVersion, int patchLevel);`

The standard distribution comes with only one version of MySQL packaged. However, it is possible to package multiple versions, and specify which version to use.

23.4.5. Connector/MXJ Notes and Tips

This section contains notes and tips on using the Connector/MXJ component within your applications.

23.4.5.1. Creating your own Connector/MXJ Package

If you want to create a custom Connector/MXJ package that includes a specific `mysqlD` version or platform then you must extract and rebuild the `connector-mxj.jar` file.

First, you should create a new directory into which you can extract the current `connector-mxj.jar`:

```
shell> mkdir custom-mxj
shell> cd custom-mxj
shell> jar -xf connector-mxj.jar
shell> ls
5-0-22/
ConnectorMXJObjectTestExample.class
ConnectorMXJUrlTestExample.class
META-INF/
TestDb.class
com/
kill.exe
```

The MySQL version directory, `5-0-22` in the above example, contains all of the files used to create an instance of MySQL when Connector/MXJ is executed. All of the files in this directory are required for each version of MySQL that you want to embed. Note as well the format of the version number, which uses hyphens instead of periods to separate the version number components.

Within the version specific directory are the platform specific directories, and

archives of the data and share directory required by MySQL for the various platforms. For example, here is the listing for the default Connector/MXJ package:

```
shell>> ls
Linux-i386/
META-INF/
Mac_OS_X-ppc/
SunOS-sparc/
Win-x86/
com/
data_dir.jar
share_dir.jar
win_share_dir.jar
```

Platform specific directories are listed by their OS and platform - for example the `mysqlld` for Mac OS X PowerPC is located within the `Mac_OS_X-ppc` directory. You can delete directories from this location that you do not require, and add new directories for additional platforms that you want to support.

To add a platform specific `mysqlld`, create a new directory with the corresponding name for your operating system/platform. For example, you could add a directory for Mac OS X/Intel using the directory `Mac_OS_X-i386`.

On Unix systems, you can determine the platform using `uname`:

```
shell> uname -p
i386
```

Now you need to download or compile `mysqlld` for the MySQL version and platform you want to include in your custom `connector-mxj.jar` package into the new directory.

Create a file called `version.txt` in the OS/platform directory you have just created that contains the version string/path of the `mysqlld` binary. For example:

```
mysql-5.0.22-osx10.3-i386/bin/mysqlld
```

You can now recreate the `connector-mxj.jar` file with the added `mysqlld`:

```
shell> cd custom-mxj
shell> jar -cf ../connector-mxj.jar *
```

You should test this package using the steps outlined in [Section 23.4.2.3, “Connector/MXJ Quick Start Guide”](#).

23.4.5.2. Deploying Connector/MXJ with a pre-configured database

To include a pre-configured/populated database within your Connector/MXJ JAR file you must create a custom `data_dir.jar` file, as included within the main `connector-mxj.jar` file:

1. First extract the `connector-mxj.jar` file, as outlined in the previous section (see [Section 23.4.5.1, “Creating your own Connector/MXJ Package”](#)).
2. First, create your database and populate the database with the information you require in an existing instance of MySQL - including Connector/MXJ instances. Data file formats are compatible across platforms.
3. Shutdown the instance of MySQL.
4. Create a JAR file of the data directory and databases that you want to include your Connector/MXJ package. You should include the `mysql` database, which includes user authentication information, in addition to the specific databases you want to include. For example, to create a JAR of the `mysql` and `mxjtest` databases:

```
shell> jar -cf ../data_dir.jar mysql mxjtest
```

5. Copy the `data_dir.jar` file into the extracted `connector-mxj.jar` directory, and then create an archive for `connector-mxj.jar`.

Note that if you are create databases using the InnoDB engine, you must include the `ibdata.*` and `ib_logfile*` files within the `data_dir.jar` archive.

23.4.5.3. Running within a JMX Agent (custom)

As a JMX MBean, MySQL Connector/MXJ requires a JMX v1.2 compliant MBean container, such as JBoss version 4. The MBean will uses the standard JMX management APIs to present (and allow the setting of) parameters which are appropriate for that platform.

If you are not using the SUN Reference implementation of the JMX libraries, you should skip this section. Or, if you are deploying to JBoss, you also may wish to skip to the next section.

We want to see the `MysqldDynamicMBean` in action inside of a JMX agent. In the `com.mysql.management.jmx.sunri` package is a custom JMX agent with two MBeans:

1. the `MysqldDynamicMBean`, and
2. a `com.sun.jdmk.comm.HtmlAdaptorServer`, which provides a web interface for manipulating the beans inside of a JMX agent.

When this very simple agent is started, it will allow a MySQL database to be started and stopped with a web browser.

1. Complete the testing of the platform as above.
 - current JDK, JUnit, Connector/J, MySQL Connector/MXJ
 - this section *requires* the SUN reference implementation of JMX
 - PATH, JAVA_HOME, ANT_HOME, CLASSPATH

2. If not building from source, skip to next step

rebuild with the "sunri.present"

```
ant -Dsunri.present=true dist
re-run tests:
java junit.textui.TestRunner com.mysql.management.AllTestsSuite
```

3. launch the test agent from the command line:

```
java com.mysql.management.jmx.sunri.MysqldTestAgentSunHtmlAdaptc
```

4. from a browser:

```
http://localhost:9092/
```

5. under MysqldAgent,

```
select "name=mysqlld"
```
6. Observe the MBean View
7. scroll to the bottom of the screen press the startMysqld button
8. click Back to MBean View
9. scroll to the bottom of the screen press stopMysqld button
10. kill the java process running the Test Agent (jmx server)

23.4.5.4. Deployment in a standard JMX Agent environment (JBoss)

Once there is confidence that the MBean will function on the platform, deploying the MBean inside of a standard JMX Agent is the next step. Included are instructions for deploying to JBoss.

1. Ensure a current version of java development kit (v1.4.x), see above.
 - Ensure JAVA_HOME is set (JBoss requires JAVA_HOME)
 - Ensure JAVA_HOME/bin is in the PATH (You will NOT need to set your CLASSPATH, nor will you need any of the jars used in the previous tests).
2. Ensure a current version of JBoss (v4.0RC1 or better)

```
http://www.jboss.org/index.html
select "Downloads"
select "jboss-4.0.zip"
pick a mirror
unzip ~/dload/jboss-4.0.zip
create a JBOSS_HOME environment variable set to the unzipped dir
unix only:
cd $JBOSS_HOME/bin
chmod +x *.sh
```

3. Deploy (copy) the connector-mxj.jar to

`$JBOSS_HOME/server/default/lib.`

4. Deploy (copy) `mysql-connector-java-3.1.4-beta-bin.jar` to `$JBOSS_HOME/server/default/lib.`
5. Create a `mxjtest.war` directory in `$JBOSS_HOME/server/default/deploy.`
6. Deploy (copy) `index.jsp` to `$JBOSS_HOME/server/default/deploy/mxjtest.war.`
7. Create a `mysqld-service.xml` file in `$JBOSS_HOME/server/default/deploy.`

```
<?xml version="1.0" encoding="UTF-8"?>
  <server>
    <mbean code="com.mysql.management.jmx.jboss.JBossMysqldDynamic
      name="mysql:type=service,name=mysqld">
      <attribute name="datadir">/tmp/xxx_data_xxx</attribute>
      <attribute name="autostart">>true</attribute>
    </mbean>
  </server>
```

8. Start jboss:
 - on unix: **`$JBOSS_HOME/bin/run.sh`**
 - on windows: **`%JBOSS_HOME%\bin\run.bat`**

Be ready: JBoss sends a lot of output to the screen.

9. When JBoss seems to have stopped sending output to the screen, open a web browser to: `http://localhost:8080/jmx-console`
10. Scroll down to the bottom of the page in the `mysql` section, select the bulleted `mysqld` link.
11. Observe the JMX MBean View page. MySQL should already be running.
12. (If "autostart=true" was set, you may skip this step.) Scroll to the bottom of the screen. You may press the Invoke button to stop (or start) MySQL observe operation completed successfully without a return value.

Click Back to MBean View

13. To confirm MySQL is running, open a web browser to `http://localhost:8080/mxjtest/` and you should see that

```
SELECT 1
```

returned with a result of

```
1
```

14. Guided by the `$JBASS_HOME/server/default/deploy/mxjtest.war/index.jsp` you will be able to use MySQL in your Web Application. There is a test database and a root user (no password) ready to experiment with. Try creating a table, inserting some rows, and doing some selects.
15. Shut down MySQL. MySQL will be stopped automatically when JBoss is stopped, or: from the browser, scroll down to the bottom of the MBean View press the stop service Invoke button to halt the service. Observe Operation completed successfully without a return value. Using ps or task manager see that MySQL is no longer running

As of 1.0.6-beta version is the ability to have the MBean start the MySQL database upon start up. Also, we've taken advantage of the JBoss life-cycle extension methods so that the database will gracefully shut down when JBoss is shutdown.

23.4.6. Connector/MXJ Support

There are a wide variety of options available for obtaining support for using Connector/MXJ. You should contact the Connector/MXJ community for help before reporting a potential bug or problem. See [Section 23.4.6.1, “Connector/MXJ Community Support”](#).

23.4.6.1. Connector/MXJ Community Support

MySQL AB provides assistance to the user community by means of a number of mailing lists and web based forums.

You can find help and support through the [MySQL and Java](#) mailing list.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [Section 1.7.1, “MySQL Mailing Lists”](#).

Community support from experienced users is also available through the [MyODBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [Section 1.7.2, “MySQL Community Support at the MySQL Forums”](#).

23.4.6.2. How to Report Connector/MXJ Problems

If you encounter difficulties or problems with Connector/MXJ, contact the Connector/MXJ community [Section 23.4.6.1, “Connector/MXJ Community Support”](#).

If reporting a problem, you should ideally include the following information with the email:

- Operating system and version
- Connector/MXJ version
- MySQL server version
- Copies of error messages or other unexpected output
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through <http://bugs.mysql.com/>.

23.5. Connector/PHP

The PHP distribution and documentation are available from the PHP Web site. MySQL provides the `mysql` and `mysql_i` extensions for the Windows operating system for MySQL versions as of 5.0.18 on <http://dev.mysql.com/downloads/connector/php/>. You can find information why you should preferably use the extensions provided by MySQL on that page. For platforms other than Windows, you should use the `mysql` or `mysql_i` extensions shipped with the PHP sources. See [Section 22.3, “MySQL PHP API”](#).

Chapter 24. Extending MySQL

Table of Contents

[24.1. MySQL Internals](#)

[24.1.1. MySQL Threads](#)

[24.1.2. MySQL Test Suite](#)

[24.2. Adding New Functions to MySQL](#)

[24.2.1. Features of the User-Defined Function Interface](#)

[24.2.2. CREATE FUNCTION Syntax](#)

[24.2.3. DROP FUNCTION Syntax](#)

[24.2.4. Adding a New User-Defined Function](#)

[24.2.5. Adding a New Native Function](#)

[24.3. Adding New Procedures to MySQL](#)

[24.3.1. Procedure Analyse](#)

[24.3.2. Writing a Procedure](#)

24.1. MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. If you plan to contribute to MySQL development, want to have access to the bleeding-edge versions of the code, or just want to keep track of development, follow the instructions in [Section 2.9.3, “Installing from the Development Source Tree”](#). If you are interested in MySQL internals, you should also subscribe to our `internals` mailing list. This list has relatively low traffic. For details on how to subscribe, please see [Section 1.7.1, “MySQL Mailing Lists”](#). All developers at MySQL AB are on the `internals` list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

24.1.1. MySQL Threads

The MySQL server creates the following threads:

- One thread manages TCP/IP file connection requests and creates a new dedicated thread to handle the authentication and SQL statement processing for each connection. (On Unix, this thread also manages Unix socket file connection requests.) On Windows, a similar thread manages shared-memory connection requests, and on Windows NT-based systems, a thread manages named-pipe connection requests. Every client connection has its own thread, although the manager threads try to avoid creating threads by consulting the thread cache first to see whether a cached thread can be used for a new connection.
- On Windows NT, there is a named pipe handler thread that does the same work as the TCP/IP connection thread on named pipe connect requests.
- On a master replication server, slave server connections are like client connections: There is one thread per connected slave.
- On a slave replication server, an I/O thread is started to connect to the master server and read updates from it. An SQL thread is started to apply updates read from the master. These two threads run independently and can be started and stopped independently.

- The signal thread handles all signals. This thread also normally handles alarms and calls `process_alarm()` to force timeouts on connections that have been idle too long.
- If **mysqld** is compiled with `-DUSE_ALARM_THREAD`, a dedicated thread that handles alarms is created. This is only used on some systems where there are problems with `sigwait()` or if you want to use the `thr_alarm()` code in your application without a dedicated signal handling thread.
- If the server is started with the `--flush_time=val` option, a dedicated thread is created to flush all tables every *val* seconds.
- Each table for which `INSERT DELAYED` statements are issued gets its own thread.

mysqladmin processlist only shows the connection, `INSERT DELAYED`, and replication threads.

24.1.2. MySQL Test Suite

The test system that is included in Unix source and binary distributions makes it possible for users and developers to perform regression tests on the MySQL code. These tests can be run on Unix.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, operating system or library issues, and is quite thorough in testing replication. Our goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system because this ensures that all future MySQL releases work well with your applications.

The test system consists of a test language interpreter (**mysqltest**), a shell script to run all tests (**mysql-test-run**), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type **make test** from the source root directory, or change location to the `mysql-test` directory and type **./mysql-test-run**. If you have installed a binary distribution, change location to the `mysql-test` directory under the installation root directory (for example, `/usr/local/mysql/mysql-test`), and run **./mysql-**

test-run. All tests should succeed. If any do not, you should try to find out why and report the problem if it indicates a bug in MySQL. See [Section 1.8, “How to Report Bugs or Problems”](#).

If one test fails, you should run **mysql-test-run** with the `--force` option to check whether any other tests fail.

If you have a copy of **mysqld** running on the machine where you want to run the test suite, you do not have to stop it, as long as it is not using ports 9306 or 9307. If either of those ports is taken, you should edit **mysql-test-run** and change the values of the master or slave port to one that is available.

In the `mysql-test` directory, you can run an individual test case with **mysql-test-run** *test_name*.

You can use the **mysqltest** language to write your own test cases. This is documented in the MySQL Test Framework manual, available at <http://dev.mysql.com/doc/>.

If you have a question about the test suite, or have a test case to contribute, send an email message to the MySQL `internals` mailing list. See [Section 1.7.1, “MySQL Mailing Lists”](#). This list does not accept attachments, so you should FTP all the relevant files to: <ftp://ftp.mysql.com/pub/mysql/upload/>

24.2. Adding New Functions to MySQL

There are two ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as object files and then added to and removed from the server dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See [Section 24.2.2, “CREATE FUNCTION Syntax”](#).
- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the **mysqld** server and become available on a permanent basis.

Each method has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you don't need to do that.
- Native functions require you to modify a source distribution. UDFs do not. You can add UDFs to a binary MySQL distribution. No access to MySQL source is necessary.
- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as `ABS()` or `SOUNDEX()`.

Another way to add functions is by creating stored functions. These are written using SQL statements rather than by compiling object code. The syntax for writing stored functions is described in [Chapter 17, *Stored Procedures and Functions*](#).

The following sections describe features of the UDF interface, provide instructions for writing UDFs, discuss security precautions that MySQL takes to

prevent UDF misuse, and describe how to add native MySQL functions.

For example source code that illustrates how to write UDFs, take a look at the `sql/udf_example.cc` file that is provided in MySQL source distributions.

24.2.1. Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilities:

- Functions can return string, integer, or real values.
- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number and types of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.
- You can indicate that a function returns NULL or that an error occurred.

24.2.2. CREATE FUNCTION Syntax

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL|  
SONAME shared_library_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` or `CONCAT()`.

function_name is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value. As of MySQL 5.0.3, `DECIMAL` is a legal value after `RETURNS`, but currently `DECIMAL` functions return string values and should be written like `STRING` functions.

shared_library_name is the basename of the shared object file that contains the code that implements the function. The file must be located in a directory that is searched by your system's dynamic linker.

To create a function, you must have the `INSERT` privilege for the `mysql` database. This is necessary because `CREATE FUNCTION` adds a row to the `mysql.func` system table that records the function's name, type, and shared library name. If you do not have this table, you should run the **mysql_upgrade** command to create it. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start **mysqld** with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see [Section 24.2.4, “Adding a New User-Defined Function”](#). For the UDF mechanism to work, functions must be written in C or C++, your operating system must support dynamic loading and you must have compiled **mysqld** dynamically (not statically).

An `AGGREGATE` function works exactly like a native MySQL aggregate (summary) function such as `SUM` or `COUNT()`. For `AGGREGATE` to work, your `mysql.func` table must contain a type column. If your `mysql.func` table does not have this column, you should run the **mysql_upgrade** program to create it (see [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#)).

24.2.3. DROP FUNCTION Syntax

```
DROP FUNCTION function_name
```

This statement drops the user-defined function (UDF) named *function_name*.

To drop a function, you must have the `DELETE` privilege for the `mysql` database. This is because `DROP FUNCTION` removes a row from the `mysql.func` system table that records the function's name, type, and shared library name.

24.2.4. Adding a New User-Defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. The MySQL source distribution includes a file `sql/udf_example.cc` that defines 5 new functions. Consult this file to see how UDF calling conventions work.

A UDF contains code that becomes part of the running server, so when you write a UDF, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. Note that these constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to UDFs that were originally written for older servers. For information about these constraints, see [Section 2.9.2, “Typical configure Options”](#), and [Section 2.9.4, “Dealing with Problems Compiling MySQL”](#).

To be able to use UDFs, you need to link **mysqld** dynamically. Don't configure MySQL using `--with-mysqld-ldflags=-all-static`. If you want to use a UDF that needs to access symbols from **mysqld** (for example, the `metaphone` function in `sql/udf_example.cc` that uses `default_charset_info`), you must link the program with `-rdynamic` (see `man dlopen`). If you plan to use UDFs, the rule of thumb is to configure MySQL with `--with-mysqld-ldflags=-rdynamic` unless you have a very good reason not to.

If you must use a precompiled distribution of MySQL, use MySQL-Max, which contains a dynamically linked server that supports dynamic loading.

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the following discussion, the name “xxx” is used for an example function name. To distinguish between SQL and C/C++ usage, `xxx()` (uppercase) indicates an SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

The C/C++ functions that you write to implement the interface for `xxx()` are:

- `xxx()` (required)

The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here:

SQL Type	C/C++ Type
STRING	char *
INTEGER	long long
REAL	double

It is also possible to declare a `DECIMAL` function, but currently the value is returned as a string, so you should write the UDF as though it were a `STRING` function.

- `xxx_init()` (optional)

The initialization function for `xxx()`. It can be used for the following purposes:

- To check the number of arguments to `xxx()`.
- To check that the arguments are of a required type or, alternatively, to tell MySQL to coerce arguments to the types you want when the main function is called.
- To allocate any memory required by the main function.
- To specify the maximum length of the result.
- To specify (for `REAL` functions) the maximum number of decimal places in the result.
- To specify whether the result can be `NULL`.

- `xxx_deinit()` (optional)

The deinitialization function for `xxx()`. It should deallocate any memory allocated by the initialization function.

When an SQL statement invokes `xxx()`, MySQL calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, MySQL aborts the SQL statement with an error message and does not call the main or deinitialization functions. Otherwise, MySQL calls the main function `xxx()` once for each row. After all rows have been processed, MySQL calls the deinitialization function `xxx_deinit()` so that it can perform any required cleanup.

For aggregate functions that work like `SUM()`, you must also provide the following functions:

- `xxx_clear()` (required in 5.0)

Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- `xxx_add()` (required)

Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call `xxx_init()` to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the `GROUP BY` expression.
3. Call `xxx_clear()` for the first row in each new group.
4. Call `xxx_add()` for each new row that belongs in the same group.
5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.
6. Repeat 3-5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A consequence of this requirement is that you are not allowed to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

24.2.4.1. UDF Calling Sequences for Simple Functions

This section describes the different functions that you need to define when you create a simple UDF. [Section 24.2.4, “Adding a New User-Defined Function”](#), describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
           char *is_null, char *error);
```

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- `my_bool maybe_null`

`xxx_init()` should set `maybe_null` to 1 if `xxx()` can return `NULL`. The default value is 1 if any of the arguments are declared `maybe_null`.

- `unsigned int decimals`

The number of decimal digits to the right of the decimal point. The default value is the maximum number of decimal digits in the arguments passed to the main function. (For example, if the function is passed 1.34, 1.345, and

1.3, the default would be 3, because 1.345 has 3 decimal digits.

- `unsigned int max_length`

The maximum length of the result. The default `max_length` value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimal digits indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

If you want to return a blob value, you can set `max_length` to 65KB or 16MB. This memory is not allocated, but the value is used to decide which data type to use if there is a need to temporarily store the data.

- `char *ptr`

A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory among themselves. `xxx_init()` should allocate the memory and assign it to this pointer:

```
initid->ptr = allocated_memory;
```

In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

- `my_bool const_item`

`xxx_init()` should set `const_item` to 1 if `xxx()` always returns the same value and to 0 otherwise.

24.2.4.2. UDF Calling Sequences for Aggregate Functions

This section describes the different functions that you need to define when you create an aggregate UDF. [Section 24.2.4, “Adding a New User-Defined Function”](#), describes the order in which MySQL calls these functions.

- `xxx_reset()`

This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given UDF_ARGS argument as the first value in your internal summary value for the group. Declare xxx_reset() as follows:

```
char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                char *is_null, char *error);
```

xxx_reset() is not needed or used in MySQL 5.0, in which the UDF interface uses xxx_clear() instead. However, you can define both xxx_reset() and xxx_clear() if you want to have your UDF work with older versions of the server. (If you do include both functions, the xxx_reset() function in many cases can be implemented internally by calling xxx_clear() to reset all variables, and then calling xxx_add() to add the UDF_ARGS argument as the first value in the group.)

- xxx_clear()

This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare xxx_clear() as follows:

```
char *xxx_clear(UDF_INIT *initid, char *is_null, char *error);
```

is_null is set to point to CHAR(0) before calling xxx_clear().

If something went wrong, you can store a value in the variable to which the error argument points. error points to a single-byte variable, not to a string buffer.

xxx_clear() is required by MySQL 5.0.

- xxx_add()

This function is called for all rows that belong to the same group, except for the first row. You should use it to add the value in the UDF_ARGS argument to your internal summary variable.

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

The `xxx()` function for an aggregate UDF should be declared the same way as for a non-aggregate UDF. See [Section 24.2.4.1, “UDF Calling Sequences for Simple Functions”](#).

For an aggregate UDF, MySQL calls the `xxx()` function after all rows in the group have been processed. You should normally never access its `UDF_ARGS` argument here but instead return a value based on your internal summary variables.

Return value handling in `xxx()` should be done the same way as for a non-aggregate UDF. See [Section 24.2.4.4, “UDF Return Values and Error Handling”](#).

The `xxx_reset()` and `xxx_add()` functions handle their `UDF_ARGS` argument the same way as functions for non-aggregate UDFs. See [Section 24.2.4.3, “UDF Argument Processing”](#).

The pointer arguments to `is_null` and `error` are the same for all calls to `xxx_reset()`, `xxx_clear()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or whether the `xxx()` function should return `NULL`. You should not store a string into `*error`! `error` points to a single-byte variable, not to a string buffer.

`*is_null` is reset for each group (before calling `xxx_clear()`). `*error` is never reset.

If `*is_null` or `*error` are set when `xxx()` returns, MySQL returns `NULL` as the result for the group function.

24.2.4.3. UDF Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

- `unsigned int arg_count`

The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

```
if (args->arg_count != 2)
```

```

{
    strcpy(message, "XXX() requires two arguments");
    return 1;
}

```

- enum Item_result *arg_type

A pointer to an array containing the types for each argument. The possible type values are STRING_RESULT, INT_RESULT, and REAL_RESULT.

To make sure that arguments are of a given type and return an error if they are not, check the arg_type array in the initialization function. For example:

```

if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message, "XXX() requires a string and an integer");
    return 1;
}

```

As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the arg_type elements to the types you want. This causes MySQL to coerce arguments to those types for each call to xxx(). For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in xxx_init():

```

args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;

```

- char **args

args->args communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument *i*, args->args[*i*] points to the argument value. (See below for instructions on how to access the value properly.) For a non-constant argument, args->args[*i*] is 0. A constant argument is an expression that uses only constants, such as 3 or 4*7-2 or SIN(3.14). A non-constant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with non-constant arguments.

For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

Functions can refer to an argument `i` as follows:

- An argument of type `STRING_RESULT` is given as a string pointer plus a length, to allow handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. You should not assume that strings are null-terminated.
- For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a long long value:

```
long long int_val;  
int_val = *((long long*) args->args[i]);
```

- For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a double value:

```
double real_val;  
real_val = *((double*) args->args[i]);
```

- `unsigned long *lengths`

For the initialization function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

24.2.4.4. UDF Return Values and Error Handling

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the message parameter. The message is returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long` `long` and `double` functions. A string function should return a pointer to the result and set `*result` and `*length` to the contents and length of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

The `result` buffer that is passed to the `xxx()` function is 255 bytes long. If your result fits in this, you don't have to worry about memory allocation for results.

If your string function needs to return a string longer than 255 bytes, you must allocate the space for it with `malloc()` in your `xxx_init()` function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See [Section 24.2.4.1, “UDF Calling Sequences for Simple Functions”](#).

To indicate a return value of `NULL` in the main function, set `*is_null` to 1:

```
*is_null = 1;
```

To indicate an error return in the main function, set `*error` to 1:

```
*error = 1;
```

If `xxx()` sets `*error` to 1 for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `xxx()` was invoked. (`xxx()` is not even called for subsequent rows.)

24.2.4.5. Compiling and Installing User-Defined Functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `sql/udf_example.cc` that is included in the MySQL source distribution.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.cc` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it's more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP number for a hostname.
- `reverse_lookup()` returns the hostname for an IP number. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so udf_example.cc
```

If you are using `gcc`, you should be able to create `udf_example.so` with a simpler command:

```
shell> make udf_example.so
```

You can easily determine the correct compiler options for your system by running this command in the `sql` directory of your MySQL source tree:

```
shell> make udf_example.o
```

You should run a compile command similar to the one that `make` displays, except that you should remove the `-c` option near the end of the line and add `-o udf_example.so` to the end of the line. (On some systems, you may need to leave the `-c` on the command.)

After you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.cc` produces a file named something like `udf_example.so` (the exact name may vary from platform to platform). Copy this file to some directory such as `/usr/lib` that

searched by your system's dynamic (runtime) linker, or add the directory in which you placed the shared object to the linker configuration file (for example, `/etc/ld.so.conf`).

The dynamic linker name is system-specific (for example, **ld-elf.so.1** on FreeBSD, **ld.so** on Linux, or **dyld** on Mac OS X). Consult your system documentation for information about the linker name and how to configure it.

On many systems, you can also set the `LD_LIBRARY` or `LD_LIBRARY_PATH` environment variable to point at the directory where you have the files for your UDF. The `dlopen` manual page tells you which variable you should use on your system. You should set this in **mysql.server** or **mysqld_safe** startup scripts and restart **mysqld**.

On some systems, the **ldconfig** program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, you can compile user-defined functions by using the following procedure:

1. You need to obtain the BitKeeper source repository for MySQL 5.0. See [Section 2.9.3, “Installing from the Development Source Tree”](#).
2. In the source repository, look in the `VC++Files/examples/udf_example` directory. There are files named `udf_example.def`, `udf_example.dsp`, and `udf_example.dsw` there.
3. In the source repository, look in the `sql` directory. Copy the `udf_example.cc` from this directory to the `VC++Files/examples/udf_example` directory and rename the file to `udf_example.cpp`.
4. Open the `udf_example.dsw` file with Visual Studio VC++ and use it to compile the UDFs as a normal project.

After the shared object file has been installed, notify **mysqld** about the new functions with these statements:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'udf_example.s
```

```
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME 'udf_examp1
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME 'udf_examp1
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME 'udf_example.so'
mysql> CREATE FUNCTION reverse_lookup
-> RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE AGGREGATE FUNCTION avgcost
-> RETURNS REAL SONAME 'udf_example.so';
```

Functions can be deleted using DROP FUNCTION:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

The CREATE FUNCTION and DROP FUNCTION statements update the func system table in the mysql database. The function's name, type and shared library name are saved in the table. You must have the INSERT and DELETE privileges for the mysql database to create and drop functions.

You should not use CREATE FUNCTION to add a function that has previously been created. If you need to reinstall a function, you should remove it with DROP FUNCTION and then reinstall it with CREATE FUNCTION. You would need to do this, for example, if you recompile a new version of your function, so that **mysqld** gets the new version. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with CREATE FUNCTION and not removed with DROP FUNCTION. All active functions are reloaded each time the server starts, unless you start **mysqld** with the --skip-grant-tables option. In this case, UDF initialization is skipped and UDFs are unavailable.

24.2.4.6. User-Defined Function Security Precautions

MySQL takes the following measures to prevent misuse of user-defined functions.

You must have the INSERT privilege to be able to use CREATE FUNCTION and the DELETE privilege to be able to use DROP FUNCTION. This is necessary because these statements add and delete rows from the mysql.func table.

UDFs should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. As of MySQL 5.0.3, **mysqld** supports an `--allow-suspicious-udfs` option that controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared object files other than those containing legitimate UDFs. If you have older UDFs that contain only the `xxx` symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the `--allow-suspicious-udfs` option. Otherwise, you should avoid enabling this capability.

UDF object files cannot be placed in arbitrary directories. They must be located in some system directory that the dynamic linker is configured to search. To enforce this restriction and prevent attempts at specifying pathnames outside of directories searched by the dynamic linker, MySQL checks the shared object file name specified in `CREATE FUNCTION` statements for pathname delimiter characters. As of MySQL 5.0.3, MySQL also checks for pathname delimiters in filenames stored in the `mysql.func` table when it loads functions. This prevents attempts at specifying illegitimate pathnames through direct manipulation of the `mysql.func` table. For information about UDFs and the runtime linker, see [Section 24.2.4.5, “Compiling and Installing User-Defined Functions”](#).

24.2.5. Adding a New Native Function

The procedure for adding a new native function is described here. Note that you cannot add native functions to a binary distribution because the procedure involves modifying MySQL source code. You must compile MySQL yourself from a source distribution. Also note that if you migrate to another version of MySQL (for example, when a new version is released), you need to repeat the procedure with the new version.

To add a new native MySQL function, follow these steps:

1. Add one line to `lex.h` that defines the function name in the `sql_functions[]` array.
2. If the function prototype is simple (just takes zero, one, two or three arguments), you should in `lex.h` specify `SYM(FUNC_ARGN)` (where *N* is the number of arguments) as the second argument in the `sql_functions[]`

array and add a function that creates a function object in `item_create.cc`. Take a look at "ABS" and `create_funcs_abs()` for an example of this.

If the function prototype is complicated (for example, if it takes a variable number of arguments), you should add two lines to `sql_yacc.yy`. One indicates the preprocessor symbol that **yacc** should define (this should be added at the beginning of the file). Then define the function parameters and add an "item" with these parameters to the `simple_expr` parsing rule. For an example, check all occurrences of `ATAN` in `sql_yacc.yy` to see how this is done.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.
4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

```
double    Item_func_newname::val()
longlong  Item_func_newname::val_int()
String    *Item_func_newname::Str(String *str)
```

If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. You should probably also define the following object function:

```
void Item_func_newname::fix_length_and_dec()
```

This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments' `maybe_null` variable. You can take a look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe. In other words, don't use any global or static variables in the functions without protecting them with mutexes)

If you want to return `NULL`, from `::val()`, `::val_int()` or `::str()` you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are some additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the `sql_string.h` file.)
- The `::str()` function should return the string that holds the result or `(char*) 0` if the result is `NULL`.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

24.3. Adding New Procedures to MySQL

In MySQL, you can define a procedure in C++ that can access and modify the data in a query before it is sent to the client. The modification can be done on a row-by-row or GROUP BY level.

We have created an example procedure to show you what can be done.

Additionally, we recommend that you take a look at `mylua`. With this you can use the LUA language to load a procedure at runtime into **mysqld**.

24.3.1. Procedure Analyse

```
analyse([max_elements],[max_memory])
```

This procedure is defined in the `sql/sql_analyse.cc`. This examines the result from your query and returns an analysis of the results:

- *max_elements* (default 256) is the maximum number of distinct values `analyse` does notice per column. This is used by `analyse` to check whether the optimal data type should be of type `ENUM`.
- *max_memory* (default 8192) is the maximum amount of memory that `analyse` should allocate per column while trying to find all distinct values.

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements],[max_m
```

24.3.2. Writing a Procedure

For the moment, the only documentation for this is the source.

You can find all information about procedures by examining the following files:

- `sql/sql_analyse.cc`
- `sql/procedure.h`
- `sql/procedure.cc`

- `sql/sql_select.cc`

Appendix A. Problems and Common Errors

Table of Contents

- [A.1. How to Determine What Is Causing a Problem](#)
- [A.2. Common Errors When Using MySQL Programs](#)
 - [A.2.1. Access denied](#)
 - [A.2.2. Can't connect to \[local\] MySQL server](#)
 - [A.2.3. Client does not support authentication protocol](#)
 - [A.2.4. Password Fails When Entered Interactively](#)
 - [A.2.5. Host 'host name' is blocked](#)
 - [A.2.6. Too many connections](#)
 - [A.2.7. Out of memory](#)
 - [A.2.8. MySQL server has gone away](#)
 - [A.2.9. Packet too large](#)
 - [A.2.10. Communication Errors and Aborted Connections](#)
 - [A.2.11. The table is full](#)
 - [A.2.12. Can't create/write to file](#)
 - [A.2.13. Commands out of sync](#)
 - [A.2.14. Ignoring user](#)
 - [A.2.15. Table 'tbl name' doesn't exist](#)
 - [A.2.16. Can't initialize character set](#)
 - [A.2.17. File Not Found](#)
- [A.3. Installation-Related Issues](#)
 - [A.3.1. Problems Linking to the MySQL Client Library](#)
 - [A.3.2. Problems with File Permissions](#)
- [A.4. Administration-Related Issues](#)
 - [A.4.1. How to Reset the Root Password](#)
 - [A.4.2. What to Do If MySQL Keeps Crashing](#)
 - [A.4.3. How MySQL Handles a Full Disk](#)
 - [A.4.4. Where MySQL Stores Temporary Files](#)
 - [A.4.5. How to Protect or Change the MySQL Unix Socket File](#)
 - [A.4.6. Time Zone Problems](#)
- [A.5. Query-Related Issues](#)
 - [A.5.1. Case Sensitivity in Searches](#)
 - [A.5.2. Problems Using DATE Columns](#)
 - [A.5.3. Problems with NULL Values](#)

- [A.5.4. Problems with Column Aliases](#)
- [A.5.5. Rollback Failure for Non-Transactional Tables](#)
- [A.5.6. Deleting Rows from Related Tables](#)
- [A.5.7. Solving Problems with No Matching Rows](#)
- [A.5.8. Problems with Floating-Point Comparisons](#)
- [A.6. Optimizer-Related Issues](#)
- [A.7. Table Definition-Related Issues](#)
 - [A.7.1. Problems with ALTER TABLE](#)
 - [A.7.2. How to Change the Order of Columns in a Table](#)
 - [A.7.3. TEMPORARY TABLE Problems](#)
- [A.8. Known Issues in MySQL](#)
 - [A.8.1. Open Issues in MySQL](#)

This appendix lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

A.1. How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:
 - The keyboard doesn't work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light doesn't change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
 - The mouse pointer doesn't move.
 - The machine doesn't answer to a remote machine's pings.
 - Other programs that are not related to MySQL don't behave correctly.
 - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as `glibc`) are up to date.

It's always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See [Section 5.12, “MySQL Server](#)

[Logs](#)".

- If you don't think you have hardware problems, you should try to find out which program is causing problems. Try using **top**, **ps**, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use **top**, **df**, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it doesn't want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that the MySQL server or a MySQL client is causing the problem, it's time to create a bug report for our mailing list or our support team. In the bug report, try to give a very detailed description of how the system is behaving and what you think is happening. You should also state why you think that MySQL is causing the problem. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the "copy and paste" method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only "the system doesn't work." This doesn't provide us with any information about what could be the problem.

If a program fails, it's always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with **top**. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the **mysqld** server is causing problems, can you get any response from it with **mysqladmin -u root ping** or **mysqladmin -u root processlist**?
- What does a client program say when you try to connect to the MySQL

server? (Try with **mysql**, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in [Section 1.8, “How to Report Bugs or Problems”](#).

A.2. Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

A.2.1. Access denied

An Access denied error can have many causes. Often the problem is related to the MySQL accounts that the server allows client programs to use when connecting. See [Section 5.8.8, “Causes of Access denied Errors”](#), and [Section 5.8.2, “How the Privilege System Works”](#).

A.2.2. Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the **mysqld** server in two different ways: By using a Unix socket file to connect through a file in the filesystem (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you don't specify a hostname or if you specify the special hostname `localhost`.

If the MySQL server is running on Windows 9x or Me, you can connect only via TCP/IP. If the server is running on Windows NT, 2000, XP, or 2003 and is started with the `--enable-named-pipe` option, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you don't give a hostname when connecting to **mysqld**, a MySQL client first tries to connect to the named pipe. If that doesn't work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the hostname.

The error (2002) Can't connect to ... normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket filename or TCP/IP port number when trying to connect to the server.

The error (2003) Can't connect to MySQL server on 'server' (10061) indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, the network port you specified is the one configured on the server, and that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

Start by checking whether there is a process named **mysqld** running on your server host. (Use **ps xa | grep mysqld** on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See [Section 2.10.2.3, “Starting and Troubleshooting the MySQL Server”](#).

If a **mysqld** process is running, you can check it by trying the following commands. The port number or Unix socket filename might be different in your setup. `host_ip` represents the IP number of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=socket --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotes with the `hostname` command; these cause the output of `hostname` (that is, the current hostname) to be substituted into the **mysqladmin** command. If you have no `hostname` command or are running on Windows, you can manually type the hostname of your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Here are some reasons the Can't connect to local MySQL server error might occur:

- **mysqld** is not running. Check your operating system's process list to ensure the **mysqld** process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: [Section A.2.2.1, “Connection to MySQL Server Failing on Windows”](#).

- You are running on a system that uses MIT-pthreads. If you are running on a system that doesn't have native threads, **mysqld** uses the MIT-pthreads package. See [Section 2.1.1, “Operating Systems Supported by MySQL”](#). However, not all MIT-pthreads versions support Unix socket files. On a system without socket file support, you must always specify the hostname explicitly when connecting to the server. Try using this command to check the connection to the server:

```
shell> mysqladmin -h `hostname` version
```

- Someone has removed the Unix socket file that **mysqld** uses (/tmp/mysql.sock by default). For example, you might have a **cron** job that removes old files from the /tmp directory. You can always run **mysqladmin version** to check whether the Unix socket file that **mysqladmin** is trying to use really exists. The fix in this case is to change the **cron** job to not remove mysql.sock or to place the socket file somewhere else. See [Section A.4.5, “How to Protect or Change the MySQL Unix Socket File”](#).
- You have started the **mysqld** server with the --socket=/path/to/socket option, but forgotten to tell client programs the new name of the socket file. If you change the socket pathname for the server, you must also notify the MySQL clients. You can do this by providing the same --socket option when you run client programs. You also need to ensure that clients have permission to access the mysql.sock file. To find out where the socket file is, you can do:

```
shell> netstat -ln | grep mysql
```

See [Section A.4.5, “How to Protect or Change the MySQL Unix Socket File”](#).

- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other **mysqld** threads (for example, with **kill** or with the **mysql_zap** script) before you can restart the MySQL server. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#).
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or

socket file so that the server and clients can access them, or restart **mysqld** with a `--socket` option that specifies a socket filename in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using **mysqladmin -h localhost variables** to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the port variable.)
- Make sure that your **mysqld** server was not started with the `--skip-networking` option. If it was, you cannot connect to it using TCP/IP.
- Check to make sure that there is no firewall blocking access to MySQL. Applications such as ZoneAlarm and the Windows XP personal firewall may need to be configured to allow external access to a MySQL server.

A.2.2.1. Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows doesn't allow for enough ephemeral (short-lived) ports to serve those connections.

By default, Windows allows 5000 ephemeral (short-lived) TCP ports to the user. After any port is closed it will remain in a `TIME_WAIT` status for 120 seconds. This status allows the connection to be reused at a much lower cost than reinitializing a brand new connection. However, the port will not be available

again until this time expires.

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the TIME_WAIT status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

IMPORTANT: The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore the registry if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base:

<http://support.microsoft.com/kb/256986/EN-US/>.

1. Start Registry Editor (Regedt32.exe).
2. Locate the following key in the registry:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Param

3. On the Edit menu, click Add Value, and then add the following registry value:

Value Name: MaxUserPort
Data Type: REG_DWORD
Value: 65534

This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the Edit menu, click Add Value, and then add the following registry value:

Value Name: TcpTimedWaitDelay
Data Type: REG_DWORD
Value: 30

This sets the number of seconds to hold a TCP port connection in TIME_WAIT state before closing. The valid range is between 0 (zero) and 300 (decimal). The default value is 0x78 (120 decimal).

5. Quit Registry Editor.
6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

A.2.3. client does not support authentication protocol

MySQL 5.0 uses an authentication protocol based on a password hashing algorithm that is incompatible with that used by older (pre-4.1) clients. If you upgrade the server from 4.0, attempts to connect to it with an older client may fail with the following message:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

To solve this problem, you should use one of the following approaches:

- Upgrade all client programs to use a 4.1.1 or newer client library.
- When connecting to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the SET PASSWORD statement and the OLD_PASSWORD() function:

```
mysql> SET PASSWORD FOR
-> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
```

Alternatively, use UPDATE and FLUSH PRIVILEGES:

```
mysql> UPDATE mysql.user SET Password = OLD_PASSWORD('newpwd')
-> WHERE Host = 'some_host' AND User = 'some_user';
mysql> FLUSH PRIVILEGES;
```

Substitute the password you want to use for “*newpwd*” in the preceding examples. MySQL cannot tell you what the original password was, so you'll need to pick a new one.

- Tell the server to use the older password hashing algorithm:
 1. Start **mysqld** with the `--old-passwords` option.
 2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

```
mysql> SELECT Host, User, Password FROM mysql.user  
-> WHERE LENGTH(Password) > 16;
```

For each account record displayed by the query, use the `Host` and `User` values and assign a password using the `OLD_PASSWORD()` function and either `SET PASSWORD` or `UPDATE`, as described earlier.

Note: In older versions of PHP, the `mysql` extension does not support the authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the `mysql` extension with MySQL 4.1 or newer, you may need to follow one of the options discussed above for configuring MySQL to work with old clients. The `mysql_i` extension (stands for “MySQL, Improved”; added in PHP 5) is compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done to use this MySQL client library. For more information about the `mysql_i` extension, see http://php.net/mysql_i.

It may also be possible to compile the older `mysql` extension against the new MySQL client library. This is beyond the scope of this Manual; consult the PHP documentation for more information. You also be able to obtain assistance with these issues in our [MySQL with PHP forum](#).

For additional background on password hashing and authentication, see [Section 5.8.9, “Password Hashing as of MySQL 4.1”](#).

A.2.4. Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--`

password or -p option that has no following password value:

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the Enter password: prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

A.2.5. Host 'host_name' is blocked

If you get the following error, it means that **mysqld** has received many connect requests from the host 'host_name' that have been interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The number of interrupted connect requests allowed is determined by the value of the `max_connect_errors` system variable. After `max_connect_errors` failed requests, **mysqld** assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you execute a **mysqladmin flush-hosts** command or issue a `FLUSH HOSTS` statement. See [Section 5.2.2, “Server System Variables”](#).

By default, **mysqld** blocks a host after 10 connection errors. You can adjust the value by starting the server like this:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

If you get this error message for a given host, you should first verify that there isn't anything wrong with TCP/IP connections from that host. If you are having network problems, it does you no good to increase the value of the `max_connect_errors` variable.

A.2.6. Too many connections

If you get a Too many connections error when you try to connect to the **mysqld** server, this means that all available connections are in use by other clients.

The number of connections allowed is controlled by the `max_connections` system variable. Its default value is 100. If you need to support more connections, you should restart **mysqld** with a larger value for this variable.

mysqld actually allows `max_connections+1` clients to connect. The extra connection is reserved for use by accounts that have the SUPER privilege. By granting the SUPER privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See [Section 13.5.4.19, “SHOW PROCESSLIST Syntax”](#).

The maximum number of connections MySQL can support depends on the quality of the thread library on a given platform. Linux or Solaris should be able to support 500-1000 simultaneous connections, depending on how much RAM you have and what your clients are doing. Static Linux binaries provided by MySQL AB can support up to 4000 connections.

A.2.7. Out of memory

If you issue a query using the **mysql** client program and receive an error like the following one, it means that **mysql** does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'  
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)  
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke **mysql** with the `--quick` option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

A.2.8. MySQL server has gone away

This section also covers the related `Lost connection to server during query`

error.

The most common reason for the MySQL server has gone away error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent):

Error Code	Description
CR_SERVER_GONE_ERROR	The client couldn't send a question to the server.
CR_SERVER_LOST	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` variable when you start **mysqld**. See [Section 5.2.2, “Server System Variables”](#).

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the MySQL server has gone away error are:

- You (or the db administrator) has killed the running thread with a `KILL` statement or a **mysqladmin kill** command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT, ...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT, ...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL`

structure is equal to 0).

- You are using a Windows client and the server had dropped the connection (probably because `wait_timeout` expired) before the command was issued.

The problem on Windows is that in some cases MySQL doesn't get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from connection.

Prior to MySQL 5.0.19, even if the `reconnect` flag in the `MYSQL` structure is equal to 1, MySQL does not automatically reconnect and re-issue the query as it doesn't know if the server did get the original query or not.

The solution to this is to either do a `mysql_ping` on the connection if there has been a long time since the last query (this is what `MyODBC` does) or set `wait_timeout` on the **mysqld** server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If **mysqld** receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by setting the server's `max_allowed_packet` variable, which has a default value of 1MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in [Section A.2.9, "Packet too large"](#).
- You also get a lost connection if you are sending a packet 16MB or larger if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.
- It is also possible to see this error if hostname lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working — from MySQL's point of view the problem is indistinguishable from any other network timeout.

You may also see the MySQL server has gone away error if MySQL is started with the `--skip-networking` option.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.

Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing **mysqladmin version** and examining the server's uptime. If the client connection was broken because **mysqld** crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#).

You can get more information about the lost connections by starting **mysqld** with the `--log-warnings=2` option. This logs some of the disconnected errors in the `hostname.err` file. See [Section 5.12.1, “The Error Log”](#).

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#).
- If a specific query kills **mysqld** and the tables involved were checked with `CHECK TABLE` before you ran the query, can you provide a reproducible test case? See [Section E.1.6, “Making a Test Case If You Experience Table Corruption”](#).
- What is the value of the `wait_timeout` system variable in the MySQL server? (**mysqladmin variables** gives you the value of this variable.)
- Have you tried to run **mysqld** with the `--log` option to determine whether the problem query appears in the log?

See also [Section A.2.10, “Communication Errors and Aborted Connections”](#), and [Section 1.8, “How to Report Bugs or Problems”](#).

A.2.9. Packet too large

A communication packet is a single SQL statement sent to the MySQL server or a single row that is sent to the client.

The largest possible packet that can be transmitted to or from a MySQL 5.0 server or client is 1GB.

When a MySQL client or the **mysqld** server receives a packet bigger than `max_allowed_packet` bytes, it issues a `Packet too large` error and closes the connection. With some clients, you may also get a `Lost connection to MySQL server during query` error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the **mysql** client program, its default `max_allowed_packet` variable is 16MB. To set a larger value, start **mysql** like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` value is 1MB. You can increase this if the server needs to handle big queries (for example, if you are working with big BLOB columns). For example, to set the variable to 16MB, start the server like this:

```
shell> mysqld --max_allowed_packet=16M
```

You can also use an option file to set `max_allowed_packet`. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=16M
```

It is safe to increase the value of this variable because the extra memory is

allocated only when needed. For example, **mysqld** allocates more memory only when you issue a long query or when **mysqld** must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large BLOB values but have not given **mysqld** access to enough memory to handle the query. If you suspect this is the case, try adding **ulimit -d 256000** to the beginning of the **mysqld_safe** script and restarting **mysqld**.

A.2.10. Communication Errors and Aborted Connections

The server error log can be a useful source of information about connection problems. See [Section 5.12.1, “The Error Log”](#). If you start the server with the `--log-warnings` option, you might find messages like this in your error log:

```
010301 14:38:23 Aborted connection 854 to db: 'users' user: 'josh'
```

If Aborted connections messages appear in the error log, the cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See [Section 5.2.2, “Server System Variables”](#).
- The client program ended abruptly in the middle of a data transfer.

When any of these things happen, the server increments the `Aborted_clients` status variable.

The server increments the `Aborted_connects` status variable when the following things happen:

- A client doesn't have privileges to connect to a database.
- A client uses an incorrect password.

- A connection packet doesn't contain the right information.
- It takes more than `connect_timeout` seconds to get a connect packet. See [Section 5.2.2, “Server System Variables”](#).

If these kinds of things happen, it might indicate that someone is trying to break into your server!

Other reasons for problems with aborted clients or aborted connections:

- Use of Ethernet protocol with Linux, both half and full duplex. Many Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file via FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. The only solution is switching the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and testing the results to determine the best setting.
- Some problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.
- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for **mysqld**. See [Section A.2.9, “Packet too large”](#).

See also [Section A.2.8, “MySQL server has gone away”](#).

A.2.11. The table is full

There are several ways a full-table error can occur:

- You are using a MySQL server older than 3.23 and an in-memory temporary table becomes larger than `tmp_table_size` bytes. To avoid this problem, you can use the `--tmp_table_size=val` option to make **mysqld** increase the temporary table size or use the SQL option `SQL_BIG_TABLES` before you issue the problematic query. See [Section 13.5.3, “SET Syntax”](#).

You can also start **mysqld** with the `--big-tables` option. This is exactly the same as using `SQL_BIG_TABLES` for all queries.

As of MySQL 3.23, this problem should not occur. If an in-memory temporary table becomes larger than `tmp_table_size`, the server automatically converts it to a disk-based MyISAM table.

- You are using InnoDB tables and run out of room in the InnoDB tablespace. In this case, the solution is to extend the InnoDB tablespace. See [Section 14.2.7, “Adding and Removing InnoDB Data and Log Files”](#).
- You are using ISAM or MyISAM tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a MyISAM table and the space required for the table exceeds what is allowed by the internal pointer size. If you don't specify the `MAX_ROWS` table option when you create a table, MySQL uses the `mysam_data_pointer_size` system variable. From MySQL 5.0.6 on, the default value is 6 bytes, which is enough to allow 256TB of data. Before MySQL 5.0.6, the default value is 4 bytes, which is enough to allow only 4GB of data. See [Section 5.2.2, “Server System Variables”](#).

You can check the maximum data/index sizes by using this statement:

```
SHOW TABLE STATUS FROM database LIKE 'tbl_name';
```

You also can use **mysamchk -dv /path/to/table-index-file**.

If the pointer size is too small, you can fix the problem by using ALTER TABLE:

```
ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

You have to specify `AVG_ROW_LENGTH` only for tables with BLOB or TEXT columns; in this case, MySQL can't optimize the space required based only on the number of rows.

A.2.12. Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start **mysqld** with the `--tmpdir` option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See [Section 4.3.2, “Using Option Files”](#).

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` directory.

Check also the error code that you get with **perror**. One reason the server cannot write to a table is that the filesystem is full:

```
shell> perror 28
Error code 28: No space left on device
```

A.2.13. Commands out of sync

If you get `Commands out of sync; you can't run this command now` in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

A.2.14. Ignoring user

If you get the following error, it means that when **mysqld** was started or when it reloaded the grant tables, it found an account in the user table that had an

invalid password.

Found wrong password for user 'some_user'@'some_host'; ignoring user

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of **mysqld** with an old user table. You can check this by executing **mysqlshow mysql user** to see whether the Password column is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.
- The account has an old password (eight characters long) and you didn't start **mysqld** with the `--old-protocol` option. Update the account in the user table to have a new password or restart **mysqld** with the `--old-protocol` option.
- You have specified a password in the user table without using the `PASSWORD()` function. Use **mysql** to update the account in the user table with a new password, making sure to use the `PASSWORD()` function:

```
mysql> UPDATE user SET Password=PASSWORD('newpwd')
-> WHERE User='some_user' AND Host='some_host';
```

A.2.15. Table 'tbl_name' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a filesystem that has case-sensitive filenames.
- Even for filesystems that are not case sensitive, such as on Windows, all

references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See [Section 13.5.4, “SHOW Syntax”](#).

A.2.16. Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multi-byte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running **configure** with the `--with-charset=charset_name` or `--with-extra-charsets=charset_name` option. See [Section 2.9.2, “Typical configure Options”](#).

All standard MySQL binaries are compiled with `--with-extra-character-sets=complex`, which enables support for all multi-byte character sets. See [Section 5.11.1, “The Character Set Used for Data and Sorting”](#).

- The character set is a simple character set that is not compiled into **mysqld**, and the character set definition files are not in the place where the client expects to find them.

In this case, you need to use one of the following methods to solve the problem:

- Recompile the client with support for the character set. See [Section 2.9.2, “Typical configure Options”](#).
- Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
- Copy the character definition files to the path where the client expects them to be.

A.2.17. File Not Found

If you get ERROR '...' not found (errno: 23), Can't open file: ... (errno: 24), or any other error with errno 23 or errno 24 from MySQL, it means that you haven't allocated enough file descriptors for the MySQL server. You can use the **perror** utility to get a description of what the error number means:

```
shell> perror 23
Error code 23: File table overflow
shell> perror 24
Error code 24: Too many open files
shell> perror 11
Error code 11: Resource temporarily unavailable
```

The problem here is that **mysqld** is trying to keep open too many files simultaneously. You can either tell **mysqld** not to open so many files at once or increase the number of file descriptors available to **mysqld**.

To tell **mysqld** to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_cache` system variable (the default value is 64). Reducing the value of `max_connections` also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to **mysqld**, you can use the `--open-files-limit` option to **mysqld_safe** or (as of MySQL 3.23.30) set the `open_files_limit` system variable. See [Section 5.2.2, “Server System Variables”](#). The easiest way to set these values is to add an option to your option file. See [Section 4.3.2, “Using Option Files”](#). If you have an old version of **mysqld** that doesn't support setting the open files limit, you can edit the **mysqld_safe** script. There is a commented-out line **ulimit -n 256** in the script. You can remove the '#' character to uncomment this line, and change the number 256 to set the number of file descriptors to be made available to **mysqld**.

`--open-files-limit` and **ulimit** can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a “hard” limit that can be overridden only if you start **mysqld_safe** or **mysqld** as root (just remember that you also need to start the server with the `--user` option in this case so that it does not continue to run as root after it starts up). If you need to increase the operating system limit on the number of file descriptors available to

each process, consult the documentation for your system.

Note: If you run the **tcs** shell, **ulimit** does not work! **tcs** also reports incorrect values when you ask for the current limits. In this case, you should start **mysqld_safe** using **sh**.

A.3. Installation-Related Issues

A.3.1. Problems Linking to the MySQL Client Library

When you are linking an application program to use the MySQL client library, you might get undefined reference errors for symbols that start with `mysql_`, such as those shown here:

```
/tmp/ccFKsdPa.o: In function `main':  
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'  
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_conn  
/tmp/ccFKsdPa.o(.text+0x57): undefined reference to `mysql_real_conn  
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'  
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'
```

You should be able to solve this problem by adding `-Ldir_path -lmysqlclient` at the end of your link command, where `dir_path` represents the pathname of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from **mysql_config** might indicate other libraries that should be specified on the link command as well.

If you get undefined reference errors for the `uncompress` or `compress` function, add `-lz` to the end of your link command and try again.

If you get undefined reference errors for a function that should exist on your system, such as `connect`, check the manual page for the function in question to determine which libraries you should add to the link command.

You might get undefined reference errors such as the following for functions that don't exist on your system:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

This usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours. In this case, you should download the latest MySQL source distribution and compile MySQL yourself. See [Section 2.9](#),

[“MySQL Installation Using a Source Distribution”](#).

You might get undefined reference errors at runtime when you try to execute a MySQL program. If these errors specify symbols that start with `mysql_` or indicate that the `mysqlclient` library can't be found, it means that your system can't find the shared `libmysqlclient.so` library. The fix for this is to tell your system to search for shared libraries where the library is located. Use whichever of the following methods is appropriate for your system:

- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY_PATH` environment variable.
- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY` environment variable.
- Copy `libmysqlclient.so` to some directory that is searched by your system, such as `/lib`, and update the shared library information by executing `ldconfig`.

Another way to solve this problem is by linking your program statically with the `-static` option, or by removing the dynamic MySQL libraries before linking your code. Before trying the second method, you should be sure that no other programs are using the dynamic libraries.

A.3.2. Problems with File Permissions

If you have problems with file permissions, the `UMASK` environment variable might be set incorrectly when **mysqld** starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default `UMASK` value is `0660`. You can change this behavior by starting **mysqld_safe** as follows:

```
shell> UMASK=384 # = 600 in octal  
shell> export UMASK  
shell> mysqld_safe &
```

By default, MySQL creates database and RAID directories with an access permission value of `0700`. You can modify this behavior by setting the

UMASK_DIR variable. If you set its value, new directories are created with the combined UMASK and UMASK_DIR values. For example, if you want to give group access to all new directories, you can do this:

```
shell> UMASK_DIR=504 # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

In MySQL 3.23.25 and above, MySQL assumes that the value for UMASK and UMASK_DIR is in octal if it starts with a zero.

See [Appendix F, Environment Variables](#).

A.4. Administration-Related Issues

A.4.1. How to Reset the Root Password

If you have never set a root password for MySQL, the server does not require a password at all for connecting as root. However, it is recommended to set a password for each account. See [Section 5.7.1, “General Security Guidelines”](#).

If you set a root password previously, but have forgotten what it was, you can set a new password. The following procedure is for Windows systems. The procedure for Unix systems is given later in this section.

The procedure under Windows:

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager:

Start Menu -> Control Panel -> Administrative Tools -> Services

Then find the MySQL service in the list, and stop it.

If your server is not running as a service, you may need to use the Task Manager to force it to stop.

3. Create a text file and place the following command within it on a single line:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPassword');
```

Save the file with any name. For this example the file will be c:\mysql-init.txt.

4. Open a console window to get to the DOS command prompt:

Start Menu -> Run -> cmd

5. We are assuming that you installed MySQL to c:\mysql. If you installed

MySQL to another location, adjust the following commands accordingly.

At the DOS command prompt, execute this command:

```
C:\> C:\mysql\bin\mysqld-nt --init-file=C:\mysql-init.txt
```

The contents of the file named by the `--init-file` option are executed at server startup, changing the root password. After the server has started successfully, you should delete `C:\mysql-init.txt`.

If you install MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld-nt.exe"  
      --defaults-file="C:\Program Files\MySQL\MySQL Server 5.  
      --init-file=C:\mysql-init.txt
```

The appropriate `--defaults-file` setting can be found using the Services Manager:

```
Start Menu -> Control Panel -> Administrative Tools -> Services
```

Find the MySQL service in the list, right-click on it, and choose the Properties option. The Path to executable field contains the `--defaults-file` setting.

6. Stop the MySQL server, then restart it in normal mode again. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.
7. You should be able to connect using the new password.

In a Unix environment, the procedure for resetting the root password is as follows:

1. Log on to your system as either the Unix root user or as the same user that the **mysqld** server runs as.
2. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, hostname, and configuration. Common locations are `/var/lib/mysql/`,

`/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the filename has the extension of `.pid` and begins with either `mysqld` or your system's hostname.

You can stop the MySQL server by sending a normal `kill` (not `kill -9`) to the **mysqld** process, using the pathname of the `.pid` file in the following command:

```
shell> kill `cat /mysql-data-directory/host_name.pid`
```

Note the use of backticks rather than forward quotes with the `cat` command; these cause the output of `cat` to be substituted into the `kill` command.

3. Create a text file and place the following command within it on a single line:

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPassword');
```

Save the file with any name. For this example the file will be `~/mysql-init`.

4. Restart the MySQL server with the special `--init-file=~/mysql-init` option:

```
shell> mysqld_safe --init-file=~/mysql-init &
```

The contents of the init-file are executed at server startup, changing the root password. After the server has started successfully you should delete `~/mysql-init`.

5. You should be able to connect using the new password.

Alternatively, on any platform, you can set the new password using the **mysql** client (but this approach is less secure):

1. Stop **mysqld** and restart it with the `--skip-grant-tables --user=root` options (Windows users omit the `--user=root` portion).
2. Connect to the **mysqld** server with this command:

```
shell> mysql -u root
```

3. Issue the following statements in the **mysql** client:

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpwd')  
-> WHERE User='root';  
mysql> FLUSH PRIVILEGES;
```

Replace “*newpwd*” with the actual root password that you want to use.

4. You should be able to connect using the new password.

A.4.2. What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This doesn't mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the **mysqld** server dies or whether your problem has to do with your client. You can check how long your **mysqld** server has been up by executing **mysqladmin version**. If **mysqld** has died and restarted, you may find the reason by looking in the server's error log. See [Section 5.12.1, “The Error Log”](#).

On some systems, you can find in the error log a stack trace of where **mysqld** died that you can resolve with the `resolve_stack_dump` program. See [Section E.1.4, “Using a Stack Trace”](#). Note that the variable values written in the error log may not always be 100% correct.

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with `--delay-key-write`, in which case data files are written but not index files.) This means that data file contents are safe even if **mysqld** crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting **mysqld** with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in **mysqld** that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as **mysqld** without locking the table properly.
- You are running many **mysqld** servers using the same data directory on a system that doesn't support good filesystem locks (normally handled by the lockd lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused **mysqld**.
- You have found a bug in the data storage code. This isn't likely, but it's at least possible. In this case, you can try to change the storage engine to another engine by using ALTER TABLE on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Please try the following things:

- Stop the **mysqld** server with **mysqladmin shutdown**, run **myisamchk --silent --force */*.MYI** from the data directory to check all MyISAM tables, and restart **mysqld**. This ensures that you are running from a clean state. See [Chapter 5, Database Administration](#).
- Start **mysqld** with the `--log` option and try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See [Section 5.12.2, “The General Query Log”](#). If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have been able to locate the bug and should submit a bug report for it. See [Section 1.8, “How to Report Bugs or](#)

[Problems](#)

- Try to make a test case that we can use to repeat the problem. See [Section E.1.6, “Making a Test Case If You Experience Table Corruption”](#).
- Try running the tests in the `mysql-test` directory and the MySQL benchmarks. See [Section 24.1.2, “MySQL Test Suite”](#). They should test MySQL rather well. You can also add code to the benchmarks that simulates your application. The benchmarks can be found in the `sql-bench` directory in a source distribution or, for a binary distribution, in the `sql-bench` directory under your MySQL installation directory.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- If you configure MySQL for debugging, it is much easier to gather information about possible errors if something goes wrong. Configuring MySQL for debugging causes a safe memory allocator to be included that can find some errors. It also provides a lot of output about what is happening. Reconfigure MySQL with the `--with-debug` or `--with-debug=full` option to **configure** and then recompile. See [Section E.1, “Debugging a MySQL Server”](#).
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to **mysqld**. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells **mysqld** not to use external locking. (This means that you cannot run two **mysqld** servers on the same data directory and that you must be careful if you use **myisamchk**. Nevertheless, it may be instructive to try the option as a test.)
- Have you tried **mysqladmin -u root processlist** when **mysqld** appears to be running but not responding? Sometimes **mysqld** is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. **mysqladmin -u root processlist** usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.

- Run the command **mysqladmin -i 5 status** or **mysqladmin -i 5 -r status** in a separate window to produce statistics while you run your other queries.
- Try the following:
 1. Start **mysqld** from **gdb** (or another debugger). See [Section E.1.3, “Debugging **mysqld** under **gdb**”](#).
 2. Run your test scripts.
 3. Print the backtrace and the local variables at the three lowest levels. In **gdb**, you can do this with the following commands when **mysqld** has crashed inside **gdb**:

```
backtrace
info local
up
info local
up
info local
```

With **gdb**, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where *N* is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See [Section 1.8, “How to Report Bugs or Problems”](#). Be even more detailed than usual. Because MySQL works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only VARCHAR columns (not BLOB or TEXT columns), you can try to change all VARCHAR to CHAR with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

The current dynamic row code has been in use at MySQL AB for several

years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.

- Do not rule out your server hardware when diagnosing problems. Defective hardware can be the cause of data corruption. Particular attention should be paid to both RAMS and hard-drives when troubleshooting hardware.

A.4.3. How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This section is relevant for writes to MyISAM tables. It also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must use **mysqladmin kill**. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition allows the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA`

INFILE or after an ALTER TABLE statement. All of these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for ALTER TABLE, the old table is left unchanged.

A.4.4. Where MySQL Stores Temporary Files

MySQL uses the value of the TMPDIR environment variable as the pathname of the directory in which to store temporary files. If you don't have TMPDIR set, MySQL uses the system default, which is normally /tmp, /var/tmp, or /usr/tmp. If the filesystem containing your temporary file directory is too small, you can use the --tmpdir option to **mysqld** to specify a directory in a filesystem where you have enough space.

In MySQL 5.0, the --tmpdir option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (':') on Unix and semicolon characters(';') on Windows, NetWare, and OS/2. **Note:** To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you should not set --tmpdir to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or LOAD DATA INFILE operations. If files in the temporary file directory are lost when the server restarts, replication fails.

MySQL creates all temporary files as hidden files. This ensures that the temporary files are removed if **mysqld** is terminated. The disadvantage of using hidden files is that you do not see a big temporary file that fills up the filesystem in which the temporary file directory is located.

When sorting (ORDER BY or GROUP BY), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))  
* number of matched rows  
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some SELECT queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form SQL_*

ALTER TABLE creates a temporary table in the same directory as the original table.

A.4.5. How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is /tmp/mysql.sock. (For some distribution formats, the directory might be different, such as /var/lib/mysql for RPMs.)

On some versions of Unix, anyone can delete files in the /tmp directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your /tmp directory so that files can be deleted only by their owners or the superuser (root). To do this, set the sticky bit on the /tmp directory by logging in as root and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the sticky bit is set by executing `ls -ld /tmp`. If the last permission character is t, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in /etc/my.cnf:

```
[mysqld]
socket=/path/to/socket
```

```
[client]
socket=/path/to/socket
```

See [Section 4.3.2, “Using Option Files”](#).

- Specify a `--socket` option on the command line to **mysqld_safe** and when you run client programs.
- Set the `MYSQL_UNIX_PORT` environment variable to the path of the Unix socket file.
- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the `--with-unix-socket-path` option when you run **configure**. See [Section 2.9.2, “Typical configure Options”](#).

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

A.4.6. Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if `UNIX_TIMESTAMP()` returns the wrong value. This should be done for the environment in which the server runs; for example, in **mysqld_safe** or **mysql.server**. See [Appendix F, *Environment Variables*](#).

You can set the time zone for the server with the `--timezone=timezone_name` option to **mysqld_safe**. You can also set it by setting the `TZ` environment variable before you start **mysqld**.

The allowable values for `--timezone` or `TZ` are system-dependent. Consult your operating system documentation to see what values are acceptable.

A.5. Query-Related Issues

A.5.1. Case Sensitivity in Searches

By default, MySQL searches are not case sensitive (although there are some character sets that are never case insensitive, such as czech). This means that if you search with `col_name LIKE 'a%'`, you get all column values that start with A or a. If you want to make this search case sensitive, make sure that one of the operands has a case sensitive or binary collation. For example, if you are comparing a column and a string that both have the `latin1` character set, you can use the `COLLATE` operator to cause either operand to have the `latin1_general_cs` or `latin1_bin` collation. For example:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation. See [Section 13.1.5, “CREATE TABLE Syntax”](#).

Simple comparison operations (`>=`, `>`, `=`, `<`, `<=`, sorting, and grouping) are based on each character's “sort value.” Characters with the same sort value (such as ‘E’, ‘e’, and ‘Ě’ (U+00c8)) are treated as the same character.

A.5.2. Problems Using DATE Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is allowed. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
mysql> SELECT * FROM tbl_name WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context (and vice versa). It is also smart enough to allow a “relaxed” string form when updating and in a `WHERE` clause that compares a date to a `TIMESTAMP`, `DATE`, or `DATETIME` column. (“Relaxed form” means that any punctuation character may be used as the separator between parts. For example,

'2004-08-15' and '2004#08#15' are equivalent.) MySQL can also convert a string containing no separators (such as '20040815'), provided it makes sense as a date.

When you compare a DATE, TIME, DATETIME, or TIMESTAMP to a constant string with the <, <=, =, >=, >, or BETWEEN operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns
- When you compare a DATE, TIME, DATETIME, or TIMESTAMP column to an expression
- When you use any other comparison method than those just listed, such as IN or STRCMP().

For these exceptional cases, the comparison is done by converting the objects to strings and performing a string comparison.

To keep things safe, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special date '0000-00-00' can be stored and retrieved as '0000-00-00'. When using a '0000-00-00' date through MyODBC, it is automatically converted to NULL in MyODBC 2.50.12 and above, because ODBC can't handle this kind of date.

Because MySQL performs the conversions described above, the following statements work:

```
mysql> INSERT INTO tbl_name (idate) VALUES (19970505);
mysql> INSERT INTO tbl_name (idate) VALUES ('19970505');
mysql> INSERT INTO tbl_name (idate) VALUES ('97-05-05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997.05.05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997 05 05');
mysql> INSERT INTO tbl_name (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM tbl_name WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT MOD(idate,100) FROM tbl_name WHERE idate >= 19970505;
```

```
mysql> SELECT idate FROM tbl_name WHERE idate >= '19970505';
```

However, the following does not work:

```
mysql> SELECT idate FROM tbl_name WHERE STRCMP(idate, '20030505')=0;
```

STRCMP() is a string function, so it converts idate to a string in 'YYYY-MM-DD' format and performs a string comparison. It does not convert '20030505' to the date '2003-05-05' and perform a date comparison.

If you are using the ALLOW_INVALID_DATES SQL mode, MySQL allows you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12.

This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

If you are not using the NO_ZERO_IN_DATE SQL mode, the day or month part can be zero. This is convenient if you want to store a birthdate in a DATE column and you know only part of the date.

If you are not using the NO_ZERO_DATE SQL mode, MySQL also allows you to store '0000-00-00' as a “dummy date.” This is in some cases more convenient than using NULL values.

If the date cannot be converted to any reasonable value, a 0 is stored in the DATE column, which is retrieved as '0000-00-00'. This is both a speed and a convenience issue. We believe that the database server's responsibility is to retrieve the same date you stored (even if the data was not logically correct in all cases). We think it is up to the application and not the server to check the dates.

If you want MySQL to check all dates and accept only legal dates (unless overridden by IGNORE), you should set sql_mode to "NO_ZERO_IN_DATE, NO_ZERO_DATE".

Date handling in MySQL 5.0.1 and earlier works like MySQL 5.0.2 with the ALLOW_INVALID_DATES SQL mode enabled.

A.5.3. Problems with NULL Values

The concept of the NULL value is a common source of confusion for newcomers to SQL, who often think that NULL is the same thing as an empty string ''. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);  
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the phone column, but the first inserts a NULL value and the second inserts an empty string. The meaning of the first can be regarded as “phone number is not known” and the meaning of the second can be regarded as “the person is known to have no phone, and thus no phone number.”

To help with NULL handling, you can use the IS NULL and IS NOT NULL operators and the IFNULL() function.

In SQL, the NULL value is never true in comparison to any other value, even NULL. An expression that contains NULL always produces a NULL value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return NULL:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

If you want to search for column values that are NULL, you cannot use an expr = NULL test. The following statement returns no rows, because expr = NULL is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for NULL values, you must use the IS NULL test. The following statements show how to find the NULL phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;  
mysql> SELECT * FROM my_table WHERE phone = '';
```

See [Section 3.3.4.6, “Working with NULL Values”](#), for additional information and examples.

You can add an index on a column that can have NULL values if you are using the MyISAM, InnoDB, or BDB, or MEMORY storage engine. Otherwise, you must declare an indexed column NOT NULL, and you cannot insert NULL into the column.

When reading data with `LOAD DATA INFILE`, empty or missing columns are updated with `' '`. If you want a `NULL` value in a column, you should use `\N` in the data file. The literal word “`NULL`” may also be used under some circumstances. See [Section 13.2.5, “LOAD DATA INFILE Syntax”](#).

When using `DISTINCT`, `GROUP BY`, or `ORDER BY`, all `NULL` values are regarded as equal.

When using `ORDER BY`, `NULL` values are presented first, or last if you specify `DESC` to sort in descending order.

Aggregate (summary) functions such as `COUNT()`, `MIN()`, and `SUM()` ignore `NULL` values. The exception to this is `COUNT(*)`, which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-`NULL` values in the `age` column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles `NULL` values specially. If you insert `NULL` into a `TIMESTAMP` column, the current date and time is inserted. If you insert `NULL` into an integer column that has the `AUTO_INCREMENT` attribute, the next number in the sequence is inserted.

A.5.4. Problems with Column Aliases

You can use an alias to refer to a column in `GROUP BY`, `ORDER BY`, or `HAVING` clauses. Aliases can also be used to give columns better names:

```
SELECT SQRT(a*b) AS root FROM tbl_name GROUP BY root HAVING root > 0  
SELECT id, COUNT(*) AS cnt FROM tbl_name GROUP BY id HAVING cnt > 0;  
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL doesn't allow you to refer to a column alias in a `WHERE` clause. This restriction is imposed because when the `WHERE` code is executed, the column value may not yet be determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name WHERE cnt > 0 GROUP BY id;
```

The `WHERE` statement is executed to determine which rows should be included in the `GROUP BY` part, whereas `HAVING` is used to decide which rows from the result

set should be used.

A.5.5. Rollback Failure for Non-Transactional Tables

If you receive the following message when trying to perform a ROLLBACK, it means that one or more of the tables you used in the transaction do not support transactions:

Warning: Some non-transactional changed tables couldn't be rolled ba

These non-transactional tables are not affected by the ROLLBACK statement.

If you were not deliberately mixing transactional and non-transactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your **mysqld** server (or that was disabled with a startup option). If **mysqld** doesn't support a storage engine, it instead creates the table as a MyISAM table, which is non-transactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';  
SHOW CREATE TABLE tbl_name;
```

See [Section 13.5.4.21, “SHOW TABLE STATUS Syntax”](#), and [Section 13.5.4.6, “SHOW CREATE TABLE Syntax”](#).

You can check which storage engines your **mysqld** server supports by using this statement:

```
SHOW ENGINES;
```

You can also use the following statement, and check the value of the variable that is associated with the storage engine in which you are interested:

```
SHOW VARIABLES LIKE 'have_%';
```

For example, to determine whether the InnoDB storage engine is available, check the value of the `have_innodb` variable.

See [Section 13.5.4.10, “SHOW ENGINES Syntax”](#), and [Section 13.5.4.24, “SHOW VARIABLES Syntax”](#).

A.5.6. Deleting Rows from Related Tables

If the total length of the DELETE statement for `related_table` is more than 1MB (the default value of the `max_allowed_packet` system variable), you should split it into smaller parts and execute multiple DELETE statements. You probably get the fastest DELETE by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the IN clause.

A.5.7. Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that doesn't return any rows, you should use the following procedure to find out what is wrong:

1. Test the query with EXPLAIN to check whether you can find something that is obviously wrong. See [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#).
2. Select only those columns that are used in the WHERE clause.
3. Remove one table at a time from the query until it returns some rows. If the tables are large, it's a good idea to use LIMIT 10 with the query.
4. Issue a SELECT for the column that should have matched a row against the table that was last removed from the query.
5. If you are comparing FLOAT or DOUBLE columns with numbers that have decimals, you can't use equality (=) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the FLOAT to a DOUBLE fixes this. See [Section A.5.8, “Problems with Floating-Point Comparisons”](#).

Similar problems may be encountered when comparing DECIMAL values prior to MySQL 5.0.3.

6. If you still can't figure out what's wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can

create a test file by dumping the tables with **mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql**. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your SELECT statement at the end of the file.

Verify that the test file demonstrates the problem by executing these commands:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Attach the test file to a bug report, which you can file using the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

A.5.8. Problems with Floating-Point Comparisons

Floating-point numbers sometimes cause confusion because they are approximate. That is, they are not stored as exact values inside computer architecture. What you can see on the screen usually is not the exact value of the number. The FLOAT and DOUBLE data types are such, and DECIMAL operations before MySQL 5.0.3 are approximate as well.

Prior to MySQL 5.0.3, DECIMAL columns store values with exact precision because they are represented as strings, but calculations on DECIMAL values are done using floating-point operations. As of 5.0.3, MySQL performs DECIMAL operations with a precision of 64 decimal digits, which should solve most common inaccuracy problems when it comes to DECIMAL columns. (If your server is from MySQL 5.0.3 or higher, but you have DECIMAL columns in tables that were created before 5.0.3, the old behavior still applies to those columns. To convert the tables to the newer DECIMAL format, dump them with **mysqldump** and reload them.)

The following example (for versions of MySQL older than 5.0.3) demonstrates the problem. It shows that even for older DECIMAL columns, calculations that are done using floating-point operations are subject to floating-point error. (Were you to replace the DECIMAL columns with FLOAT, similar problems would occur for all versions of MySQL.)

```
mysql> CREATE TABLE t1 (i INT, d1 DECIMAL(9,2), d2 DECIMAL(9,2));
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
```

```

-> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
-> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
-> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
-> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
-> (6, 0.00, 0.00), (6, -51.40, 0.00);

```

```

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;

```

i	a	b
1	21.40	21.40
2	76.80	76.80
3	7.40	7.40
4	15.40	15.40
5	7.20	7.20
6	-51.40	0.00

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of a and b do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

As of MySQL 5.0.3, you will get only the last row in the above result.

The problem cannot be solved by using `ROUND()` or similar functions, because the result is still a floating-point number:

```

mysql> SELECT i, ROUND(SUM(d1), 2) AS a, ROUND(SUM(d2), 2) AS b
-> FROM t1 GROUP BY i HAVING a <> b;

```

i	a	b
1	21.40	21.40
2	76.80	76.80
3	7.40	7.40
4	15.40	15.40
5	7.20	7.20
6	-51.40	0.00

This is what the numbers in column a look like when displayed with more decimal places:

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1.0000000000000000 AS a,
-> ROUND(SUM(d2), 2) AS b FROM t1 GROUP BY i HAVING a <> b;
+-----+-----+-----+
| i     | a                               | b     |
+-----+-----+-----+
| 1     | 21.3999999999999986           | 21.40 |
| 2     | 76.7999999999999972           | 76.80 |
| 3     | 7.4000000000000004           | 7.40  |
| 4     | 15.4000000000000004           | 15.40 |
| 5     | 7.2000000000000002           | 7.20  |
| 6     | -51.3999999999999986         | 0.00  |
+-----+-----+-----+
```

Depending on your computer architecture, you may or may not see similar results. For example, on some machines you may get the “correct” results by multiplying both arguments by 1, as the following example shows.

Warning: Never use this method in your applications. It is not an example of a trustworthy method!

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1 AS a, ROUND(SUM(d2), 2)*1 AS b
-> FROM t1 GROUP BY i HAVING a <> b;
+-----+-----+-----+
| i     | a     | b     |
+-----+-----+-----+
| 6     | -51.40 | 0.00  |
+-----+-----+-----+
```

The reason that the preceding example seems to work is that on the particular machine where the test was done, CPU floating-point arithmetic happens to round the numbers to the same value. However, there is no rule that any CPU should do so, so this method cannot be trusted.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) > 0.0001;
+-----+-----+-----+
| i     | a     | b     |
+-----+-----+-----+
```

```
|      6 | -51.40 | 0.00 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) <= 0.0001;
```

```
+-----+-----+-----+
| i      | a      | b      |
+-----+-----+-----+
|      1 | 21.40 | 21.40 |
|      2 | 76.80 | 76.80 |
|      3 |  7.40 |  7.40 |
|      4 | 15.40 | 15.40 |
|      5 |  7.20 |  7.20 |
+-----+-----+-----+
```

A.6. Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL doesn't have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the `EXPLAIN` statement to get information about how MySQL processes a query. To use it, just add the keyword `EXPLAIN` to the front of your `SELECT` statement:

```
mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
```

`EXPLAIN` is discussed in more detail in [Section 7.2.1, “Optimizing Queries with `EXPLAIN`”](#).

- Use `ANALYZE TABLE tbl_name` to update the key distributions for the scanned table. See [Section 13.5.2.1, “`ANALYZE TABLE` Syntax”](#).
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index. See [Section 13.2.7, “`SELECT` Syntax”](#).

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

`USE INDEX` and `IGNORE INDEX` may also be useful.

- Global and table-level `STRAIGHT_JOIN`. See [Section 13.2.7, “`SELECT` Syntax”](#).
- You can tune global or thread-specific system variables. For example, Start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See [Section 5.2.2, “Server System Variables”](#).

A.7. Table Definition-Related Issues

A.7.1. Problems with ALTER TABLE

ALTER TABLE changes a table to the current character set. If you get a duplicate-key error during ALTER TABLE, the cause is either that the new character sets maps two keys to the same value or that the table is corrupted. In the latter case, you should run REPAIR TABLE on the table.

If ALTER TABLE dies with the following error, the problem may be that MySQL crashed during an earlier ALTER TABLE operation and there is an old table named A-xxx or B-xxx lying around:

```
Error on rename of './database/name.frm'  
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with A- or B-. (You may want to move them elsewhere instead of deleting them.)

ALTER TABLE works in the following way:

- Create a new table named A-xxx with the requested structural changes.
- Copy all rows from the original table to A-xxx.
- Rename the original table to B-xxx.
- Rename A-xxx to your original table name.
- Delete B-xxx.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as B-xxx. A simple rename of the table files at the system level should get your data back.

If you use ALTER TABLE on a transactional table or if you are using Windows or OS/2, ALTER TABLE unlocks the table if you had done a LOCK TABLE on it. This is

done because InnoDB and these operating systems cannot drop a table that is in use.

A.7.2. How to Change the Order of Columns in a Table

First, consider whether you really need to change the column order in a table. The whole point of SQL is to abstract the application from the data storage format. You should always specify the order in which you wish to retrieve your data. The first of the following statements returns columns in the order *col_name1, col_name2, col_name3*, whereas the second returns them in the order *col_name1, col_name3, col_name2*:

```
mysql> SELECT col_name1, col_name2, col_name3 FROM tbl_name;  
mysql> SELECT col_name1, col_name3, col_name2 FROM tbl_name;
```

If you decide to change the order of table columns anyway, you can do so as follows:

1. Create a new table with the columns in the new order.
2. Execute this statement:

```
mysql> INSERT INTO new_table  
-> SELECT columns-in-new-order FROM old_table;
```

3. Drop or rename *old_table*.
4. Rename the new table to the original name:

```
mysql> ALTER TABLE new_table RENAME old_table;
```

`SELECT *` is quite suitable for testing queries. However, in an application, you should *never* rely on using `SELECT *` and retrieving the columns based on their position. The order and position in which columns are returned does not remain the same if you add, move, or delete columns. A simple change to your table structure could cause your application to fail.

A.7.3. TEMPORARY TABLE Problems

The following list indicates limitations on the use of TEMPORARY tables:

- A TEMPORARY table can only be of type HEAP, ISAM, MyISAM, MERGE, or InnoDB.
- You cannot refer to a TEMPORARY table more than once in the same query. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;  
ERROR 1137: Can't reopen table: 'temp_table'
```

- The SHOW TABLES statement does not list TEMPORARY tables.
- You cannot use RENAME to rename a TEMPORARY table. However, you can use ALTER TABLE instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

- There are known issues in using temporary tables with replication. See [Section 6.7, “Replication Features and Known Problems”](#), for more information.

A.8. Known Issues in MySQL

This section is a list of the known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and porting instructions in [Section 2.13, “Operating System-Specific Notes”](#), and [Appendix E, *Porting to Other Systems*](#).

A.8.1. Open Issues in MySQL

The following problems are known and fixing them is a high priority:

- If you compare a `NULL` value to a subquery using `ALL/ANY/SOME` and the subquery returns an empty result, the comparison might evaluate to the non-standard result of `NULL` rather than to `TRUE` or `FALSE`. This will be fixed in MySQL 5.1.
- Subquery optimization for `IN` is not as effective as for `=`.
- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint doesn't work in replication because the constraint may have another name on the slave.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` doesn't work inside `GROUP_CONCAT()` if you don't use all and only those columns that are in the `DISTINCT` list.
- If one user has a long-running transaction and another user drops a table that is updated in the transaction, there is small chance that the binary log may contain the `DROP TABLE` command before the table is used in the transaction itself. We plan to fix this by having the `DROP TABLE` command wait until the table is not being used in any transaction.

- When inserting a big integer value (between 2^{63} and $2^{64}-1$) into a decimal or string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- `FLUSH TABLES WITH READ LOCK` does not block `COMMIT` if the server is running without binary logging, which may cause a problem (of consistency between tables) when doing a full backup.
- `ANALYZE TABLE` on a BDB table may in some cases make the table unusable until you restart **mysqld**. If this happens, look for errors of the following form in the MySQL error file:


```
001207 22:07:56 bdb: log_flush: LSN past current end-of-log
```
- Don't execute `ALTER TABLE` on a BDB table on which you are running multiple-statement transactions until all those transactions complete. (The transaction might be ignored.)
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` may cause problems on tables for which you are using `INSERT DELAYED`.
- Performing `LOCK TABLE ...` and `FLUSH TABLES ...` doesn't guarantee that there isn't a half-finished transaction in progress on the table.
- BDB tables are relatively slow to open. If you have many BDB tables in a database, it takes a long time to use the **mysql** client on the database if you are not using the `-A` option or if you are using `rehash`. This is especially noticeable when you have a large table cache.
- Replication uses query-level logging: The master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases.

It is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is non-deterministic (generally not a recommended practice, even outside of replication).

For example:

- CREATE ... SELECT or INSERT ... SELECT statements that insert zero or NULL values into an AUTO_INCREMENT column.
- DELETE if you are deleting rows from a table that has foreign keys with ON DELETE CASCADE properties.
- REPLACE ... SELECT, INSERT IGNORE ... SELECT if you have duplicate key values in the inserted data.

If and only if the preceding queries have no ORDER BY clause guaranteeing a deterministic order.

For example, for INSERT ... SELECT with no ORDER BY, the SELECT may return rows in a different order (which results in a row having different ranks, hence getting a different number in the AUTO_INCREMENT column), depending on the choices made by the optimizers on the master and slave.

A query is optimized differently on the master and slave only if:

- The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use InnoDB on the master, but MyISAM on the slave if the slave has less available disk space.)
- MySQL buffer sizes (key_buffer_size, and so on) are different on the master and slave.
- The master and slave run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using **mysqlbinlog|mysql**.

The easiest way to avoid this problem is to add an ORDER BY clause to the aforementioned non-deterministic queries to ensure that the rows are always stored or modified in the same order.

In future MySQL versions, we will automatically add an ORDER BY clause when needed.

The following issues are known and will be fixed in due time:

- Log filenames are based on the server hostname (if you don't specify a filename with the startup option). You have to use options such as `--log-bin=old_host_name-bin` if you change your hostname to something else. Another option is to rename the old files to reflect your hostname change (if these are binary logs, you need to edit the binary log index file and fix the binlog names there as well). See [Section 5.2.1, “mysqld Command Options”](#).
- **mysqlbinlog** does not delete temporary files left after a `LOAD DATA INFILE` command. See [Section 8.10, “mysqlbinlog — Utility for Processing Binary Log Files”](#).
- `RENAME` doesn't work with `TEMPORARY` tables or tables used in a `MERGE` table.
- Due to the way table format (`.frm`) files are stored, you cannot use character 255 (`CHAR(255)`) in table names, column names, or enumerations. This is scheduled to be fixed in version 5.1 when we implement new table definition format files.
- When using `SET CHARACTER SET`, you can't use translated characters in database, table, and column names.
- You can't use `'_'` or `'%'` with `ESCAPE` in `LIKE ... ESCAPE`.
- If you have a `DECIMAL` column in which the same number is stored in different formats (for example, `+01.00`, `1.00`, `01.00`), `GROUP BY` may regard each value as a different value.
- You cannot build the server in another directory when using MIT-pthreads. Because this requires changes to MIT-pthreads, we are not likely to fix this. See [Section 2.9.5, “MIT-pthreads Notes”](#).
- `BLOB` and `TEXT` values can't reliably be used in `GROUP BY`, `ORDER BY` or `DISTINCT`. Only the first `max_sort_length` bytes are used when comparing `BLOB` values in these cases. The default value of `max_sort_length` is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64

bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with `BIGINT` precision, `IF` and `ELT()` with `BIGINT` or `DOUBLE` precision, and the rest with `DOUBLE` precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.

- You can have up to 255 `ENUM` and `SET` columns in one table.
- In `MIN()`, `MAX()`, and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set.
- **`mysqld_safe`** redirects all messages from **`mysqld`** to the **`mysqld`** log. One problem with this is that if you execute **`mysqladmin refresh`** to close and reopen the log, `stdout` and `stderr` are still redirected to the old log. If you use `--log` extensively, you should edit **`mysqld_safe`** to log to `host_name.err` instead of `host_name.log` so that you can easily reclaim the space for the old log by deleting it and executing **`mysqladmin refresh`**.
- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by 2, **not 1**:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following doesn't work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;  
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns don't participate in the `DISTINCT` comparison. We will probably change this in the future to never compare the hidden columns when executing `DISTINCT`.

An example of this is:

```
SELECT DISTINCT mp3id FROM band_downloads
      WHERE userid = 9 ORDER BY id DESC;
```

and

```
SELECT DISTINCT band_downloads.mp3id
      FROM band_downloads,band_mp3
      WHERE band_downloads.userid = 9
      AND band_mp3.id = band_downloads.mp3id
      ORDER BY band_downloads.id DESC;
```

In the second case, using MySQL Server 3.23.x, you may get two identical rows in the result set (because the values in the hidden id column may differ).

Note that this happens only for queries where that do not have the ORDER BY columns in the result.

- If you execute a PROCEDURE on a query that returns an empty set, in some cases the PROCEDURE does not transform the columns.
- Creation of a table of type MERGE doesn't check whether the underlying tables are compatible types.
- If you use ALTER TABLE to add a UNIQUE index to a table used in a MERGE table and then add a normal index on the MERGE table, the key order is different for the tables if there was an old, non-UNIQUE key in the table. This is because ALTER TABLE puts UNIQUE indexes before normal indexes to be able to detect duplicate keys as early as possible.

Appendix B. Error Codes and Messages

Table of Contents

[B.1. Server Error Codes and Messages](#)

[B.2. Client Error Codes and Messages](#)

This appendix lists the errors that may appear when you call MySQL from any host language. The first list displays server error messages. The second list displays client program messages.

B.1. Server Error Codes and Messages

Server error information comes from the following source files. For details about the way that error information is defined, see the MySQL Internals manual, available at <http://dev.mysql.com/doc/>.

- Error message information is listed in the `share/errmsg.txt` file. `%d` and `%s` represent numbers and strings, respectively, that are substituted into the Message values when they are displayed.
- The Error values listed in `share/errmsg.txt` are used to generate the definitions in the `include/mysqld_error.h` and `include/mysqld_ename.h` MySQL source files.
- The `SQLSTATE` values listed in `share/errmsg.txt` are used to generate the definitions in the `include/sql_state.h` MySQL source file.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: 1000 SQLSTATE: HY000 (ER_HASHCHK)
Message: hashchk
- Error: 1001 SQLSTATE: HY000 (ER_NISAMCHK)
Message: isamchk
- Error: 1002 SQLSTATE: HY000 (ER_NO)
Message: NO
- Error: 1003 SQLSTATE: HY000 (ER_YES)
Message: YES
- Error: 1004 SQLSTATE: HY000 (ER_CANT_CREATE_FILE)
Message: Can't create file '%s' (errno: %d)

- Error: 1005 SQLSTATE: HY000 (ER_CANT_CREATE_TABLE)
Message: Can't create table '%s' (errno: %d)
- Error: 1006 SQLSTATE: HY000 (ER_CANT_CREATE_DB)
Message: Can't create database '%s' (errno: %d)
- Error: 1007 SQLSTATE: HY000 (ER_DB_CREATE_EXISTS)
Message: Can't create database '%s'; database exists
- Error: 1008 SQLSTATE: HY000 (ER_DB_DROP_EXISTS)
Message: Can't drop database '%s'; database doesn't exist
- Error: 1009 SQLSTATE: HY000 (ER_DB_DROP_DELETE)
Message: Error dropping database (can't delete '%s', errno: %d)
- Error: 1010 SQLSTATE: HY000 (ER_DB_DROP_RMDIR)
Message: Error dropping database (can't rmdir '%s', errno: %d)
- Error: 1011 SQLSTATE: HY000 (ER_CANT_DELETE_FILE)
Message: Error on delete of '%s' (errno: %d)
- Error: 1012 SQLSTATE: HY000 (ER_CANT_FIND_SYSTEM_REC)
Message: Can't read record in system table
- Error: 1013 SQLSTATE: HY000 (ER_CANT_GET_STAT)
Message: Can't get status of '%s' (errno: %d)
- Error: 1014 SQLSTATE: HY000 (ER_CANT_GET_WD)
Message: Can't get working directory (errno: %d)
- Error: 1015 SQLSTATE: HY000 (ER_CANT_LOCK)

Message: Can't lock file (errno: %d)

- Error: 1016 SQLSTATE: HY000 (ER_CANT_OPEN_FILE)

Message: Can't open file: '%s' (errno: %d)

- Error: 1017 SQLSTATE: HY000 (ER_FILE_NOT_FOUND)

Message: Can't find file: '%s' (errno: %d)

- Error: 1018 SQLSTATE: HY000 (ER_CANT_READ_DIR)

Message: Can't read dir of '%s' (errno: %d)

- Error: 1019 SQLSTATE: HY000 (ER_CANT_SET_WD)

Message: Can't change dir to '%s' (errno: %d)

- Error: 1020 SQLSTATE: HY000 (ER_CHECKREAD)

Message: Record has changed since last read in table '%s'

- Error: 1021 SQLSTATE: HY000 (ER_DISK_FULL)

Message: Disk full (%s); waiting for someone to free some space...

- Error: 1022 SQLSTATE: 23000 (ER_DUP_KEY)

Message: Can't write; duplicate key in table '%s'

- Error: 1023 SQLSTATE: HY000 (ER_ERROR_ON_CLOSE)

Message: Error on close of '%s' (errno: %d)

- Error: 1024 SQLSTATE: HY000 (ER_ERROR_ON_READ)

Message: Error reading file '%s' (errno: %d)

- Error: 1025 SQLSTATE: HY000 (ER_ERROR_ON_RENAME)

Message: Error on rename of '%s' to '%s' (errno: %d)

- Error: 1026 SQLSTATE: HY000 (ER_ERROR_ON_WRITE)
Message: Error writing file '%s' (errno: %d)
- Error: 1027 SQLSTATE: HY000 (ER_FILE_USED)
Message: '%s' is locked against change
- Error: 1028 SQLSTATE: HY000 (ER_FILSORT_ABORT)
Message: Sort aborted
- Error: 1029 SQLSTATE: HY000 (ER_FORM_NOT_FOUND)
Message: View '%s' doesn't exist for '%s'
- Error: 1030 SQLSTATE: HY000 (ER_GET_ERRNO)
Message: Got error %d from storage engine
- Error: 1031 SQLSTATE: HY000 (ER_ILLEGAL_HA)
Message: Table storage engine for '%s' doesn't have this option
- Error: 1032 SQLSTATE: HY000 (ER_KEY_NOT_FOUND)
Message: Can't find record in '%s'
- Error: 1033 SQLSTATE: HY000 (ER_NOT_FORM_FILE)
Message: Incorrect information in file: '%s'
- Error: 1034 SQLSTATE: HY000 (ER_NOT_KEYFILE)
Message: Incorrect key file for table '%s'; try to repair it
- Error: 1035 SQLSTATE: HY000 (ER_OLD_KEYFILE)
Message: Old key file for table '%s'; repair it!
- Error: 1036 SQLSTATE: HY000 (ER_OPEN_AS_READONLY)

Message: Table '%s' is read only

- Error: 1037 SQLSTATE: HY001 (ER_OUTOFMEMORY)

Message: Out of memory; restart server and try again (needed %d bytes)

- Error: 1038 SQLSTATE: HY001 (ER_OUT_OF_SORTMEMORY)

Message: Out of sort memory; increase server sort buffer size

- Error: 1039 SQLSTATE: HY000 (ER_UNEXPECTED_EOF)

Message: Unexpected EOF found when reading file '%s' (errno: %d)

- Error: 1040 SQLSTATE: 08004 (ER_CON_COUNT_ERROR)

Message: Too many connections

- Error: 1041 SQLSTATE: HY000 (ER_OUT_OF_RESOURCES)

Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space

- Error: 1042 SQLSTATE: 08S01 (ER_BAD_HOST_ERROR)

Message: Can't get hostname for your address

- Error: 1043 SQLSTATE: 08S01 (ER_HANDSHAKE_ERROR)

Message: Bad handshake

- Error: 1044 SQLSTATE: 42000 (ER_DBACCESS_DENIED_ERROR)

Message: Access denied for user '%s'@'%s' to database '%s'

- Error: 1045 SQLSTATE: 28000 (ER_ACCESS_DENIED_ERROR)

Message: Access denied for user '%s'@'%s' (using password: %s)

- Error: 1046 SQLSTATE: 3D000 (ER_NO_DB_ERROR)

Message: No database selected

- Error: 1047 SQLSTATE: 08S01 (ER_UNKNOWN_COM_ERROR)

Message: Unknown command

- Error: 1048 SQLSTATE: 23000 (ER_BAD_NULL_ERROR)

Message: Column '%s' cannot be null

- Error: 1049 SQLSTATE: 42000 (ER_BAD_DB_ERROR)

Message: Unknown database '%s'

- Error: 1050 SQLSTATE: 42S01 (ER_TABLE_EXISTS_ERROR)

Message: Table '%s' already exists

- Error: 1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)

Message: Unknown table '%s'

- Error: 1052 SQLSTATE: 23000 (ER_NON_UNIQ_ERROR)

Message: Column '%s' in %s is ambiguous

- Error: 1053 SQLSTATE: 08S01 (ER_SERVER_SHUTDOWN)

Message: Server shutdown in progress

- Error: 1054 SQLSTATE: 42S22 (ER_BAD_FIELD_ERROR)

Message: Unknown column '%s' in '%s'

- Error: 1055 SQLSTATE: 42000 (ER_WRONG_FIELD_WITH_GROUP)

Message: '%s' isn't in GROUP BY

- Error: 1056 SQLSTATE: 42000 (ER_WRONG_GROUP_FIELD)

Message: Can't group on '%s'

- Error: 1057 SQLSTATE: 42000 (ER_WRONG_SUM_SELECT)
Message: Statement has sum functions and columns in same statement
- Error: 1058 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT)
Message: Column count doesn't match value count
- Error: 1059 SQLSTATE: 42000 (ER_TOO_LONG_IDENT)
Message: Identifier name '%s' is too long
- Error: 1060 SQLSTATE: 42S21 (ER_DUP_FIELDNAME)
Message: Duplicate column name '%s'
- Error: 1061 SQLSTATE: 42000 (ER_DUP_KEYNAME)
Message: Duplicate key name '%s'
- Error: 1062 SQLSTATE: 23000 (ER_DUP_ENTRY)
Message: Duplicate entry '%s' for key %d
- Error: 1063 SQLSTATE: 42000 (ER_WRONG_FIELD_SPEC)
Message: Incorrect column specifier for column '%s'
- Error: 1064 SQLSTATE: 42000 (ER_PARSE_ERROR)
Message: %s near '%s' at line %d
- Error: 1065 SQLSTATE: 42000 (ER_EMPTY_QUERY)
Message: Query was empty
- Error: 1066 SQLSTATE: 42000 (ER_NONUNIQ_TABLE)
Message: Not unique table/alias: '%s'
- Error: 1067 SQLSTATE: 42000 (ER_INVALID_DEFAULT)

Message: Invalid default value for '%s'

- Error: 1068 SQLSTATE: 42000 (ER_MULTIPLE_PRI_KEY)

Message: Multiple primary key defined

- Error: 1069 SQLSTATE: 42000 (ER_TOO_MANY_KEYS)

Message: Too many keys specified; max %d keys allowed

- Error: 1070 SQLSTATE: 42000 (ER_TOO_MANY_KEY_PARTS)

Message: Too many key parts specified; max %d parts allowed

- Error: 1071 SQLSTATE: 42000 (ER_TOO_LONG_KEY)

Message: Specified key was too long; max key length is %d bytes

- Error: 1072 SQLSTATE: 42000 (ER_KEY_COLUMN_DOES_NOT_EXIST)

Message: Key column '%s' doesn't exist in table

- Error: 1073 SQLSTATE: 42000 (ER_BLOB_USED_AS_KEY)

Message: BLOB column '%s' can't be used in key specification with the used table type

- Error: 1074 SQLSTATE: 42000 (ER_TOO_BIG_FIELDLENGTH)

Message: Column length too big for column '%s' (max = %d); use BLOB or TEXT instead

- Error: 1075 SQLSTATE: 42000 (ER_WRONG_AUTO_KEY)

Message: Incorrect table definition; there can be only one auto column and it must be defined as a key

- Error: 1076 SQLSTATE: HY000 (ER_READY)

Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d

- Error: 1077 SQLSTATE: HY000 (ER_NORMAL_SHUTDOWN)
Message: %s: Normal shutdown
- Error: 1078 SQLSTATE: HY000 (ER_GOT_SIGNAL)
Message: %s: Got signal %d. Aborting!
- Error: 1079 SQLSTATE: HY000 (ER_SHUTDOWN_COMPLETE)
Message: %s: Shutdown complete
- Error: 1080 SQLSTATE: 08S01 (ER_FORCING_CLOSE)
Message: %s: Forcing close of thread %ld user: '%s'
- Error: 1081 SQLSTATE: 08S01 (ER_IPSOCK_ERROR)
Message: Can't create IP socket
- Error: 1082 SQLSTATE: 42S12 (ER_NO_SUCH_INDEX)
Message: Table '%s' has no index like the one used in CREATE INDEX;
recreate the table
- Error: 1083 SQLSTATE: 42000 (ER_WRONG_FIELD_TERMINATORS)
Message: Field separator argument is not what is expected; check the
manual
- Error: 1084 SQLSTATE: 42000 (ER_BLOBS_AND_NO_TERMINATED)
Message: You can't use fixed rowlength with BLOBs; please use 'fields
terminated by'
- Error: 1085 SQLSTATE: HY000 (ER_TEXTFILE_NOT_READABLE)
Message: The file '%s' must be in the database directory or be readable by
all
- Error: 1086 SQLSTATE: HY000 (ER_FILE_EXISTS_ERROR)

Message: File '%s' already exists

- Error: 1087 SQLSTATE: HY000 (ER_LOAD_INFO)

Message: Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld

- Error: 1088 SQLSTATE: HY000 (ER_ALTER_INFO)

Message: Records: %ld Duplicates: %ld

- Error: 1089 SQLSTATE: HY000 (ER_WRONG_SUB_KEY)

Message: Incorrect sub part key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique sub keys

- Error: 1090 SQLSTATE: 42000 (ER_CANT_REMOVE_ALL_FIELDS)

Message: You can't delete all columns with ALTER TABLE; use DROP TABLE instead

- Error: 1091 SQLSTATE: 42000 (ER_CANT_DROP_FIELD_OR_KEY)

Message: Can't DROP '%s'; check that column/key exists

- Error: 1092 SQLSTATE: HY000 (ER_INSERT_INFO)

Message: Records: %ld Duplicates: %ld Warnings: %ld

- Error: 1093 SQLSTATE: HY000 (ER_UPDATE_TABLE_USED)

Message: You can't specify target table '%s' for update in FROM clause

- Error: 1094 SQLSTATE: HY000 (ER_NO_SUCH_THREAD)

Message: Unknown thread id: %lu

- Error: 1095 SQLSTATE: HY000 (ER_KILL_DENIED_ERROR)

Message: You are not owner of thread %lu

- Error: 1096 SQLSTATE: HY000 (ER_NO_TABLES_USED)
Message: No tables used
- Error: 1097 SQLSTATE: HY000 (ER_TOO_BIG_SET)
Message: Too many strings for column %s and SET
- Error: 1098 SQLSTATE: HY000 (ER_NO_UNIQUE_LOGFILE)
Message: Can't generate a unique log-filename %s.(1-999)
- Error: 1099 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED_FOR_WRITE)
Message: Table '%s' was locked with a READ lock and can't be updated
- Error: 1100 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED)
Message: Table '%s' was not locked with LOCK TABLES
- Error: 1101 SQLSTATE: 42000 (ER_BLOB_CANT_HAVE_DEFAULT)
Message: BLOB/TEXT column '%s' can't have a default value
- Error: 1102 SQLSTATE: 42000 (ER_WRONG_DB_NAME)
Message: Incorrect database name '%s'
- Error: 1103 SQLSTATE: 42000 (ER_WRONG_TABLE_NAME)
Message: Incorrect table name '%s'
- Error: 1104 SQLSTATE: 42000 (ER_TOO_BIG_SELECT)
Message: The SELECT would examine more than MAX_JOIN_SIZE rows;
check your WHERE and use SET SQL_BIG_SELECTS=1 or SET
SQL_MAX_JOIN_SIZE=# if the SELECT is okay
- Error: 1105 SQLSTATE: HY000 (ER_UNKNOWN_ERROR)
Message: Unknown error

- Error: 1106 SQLSTATE: 42000 (ER_UNKNOWN_PROCEDURE)
Message: Unknown procedure '%s'
- Error: 1107 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_PROCEDURE)
Message: Incorrect parameter count to procedure '%s'
- Error: 1108 SQLSTATE: HY000 (ER_WRONG_PARAMETERS_TO_PROCEDURE)
Message: Incorrect parameters to procedure '%s'
- Error: 1109 SQLSTATE: 42S02 (ER_UNKNOWN_TABLE)
Message: Unknown table '%s' in %s
- Error: 1110 SQLSTATE: 42000 (ER_FIELD_SPECIFIED_TWICE)
Message: Column '%s' specified twice
- Error: 1111 SQLSTATE: HY000 (ER_INVALID_GROUP_FUNC_USE)
Message: Invalid use of group function
- Error: 1112 SQLSTATE: 42000 (ER_UNSUPPORTED_EXTENSION)
Message: Table '%s' uses an extension that doesn't exist in this MySQL version
- Error: 1113 SQLSTATE: 42000 (ER_TABLE_MUST_HAVE_COLUMNS)
Message: A table must have at least 1 column
- Error: 1114 SQLSTATE: HY000 (ER_RECORD_FILE_FULL)
Message: The table '%s' is full
- Error: 1115 SQLSTATE: 42000 (ER_UNKNOWN_CHARACTER_SET)
Message: Unknown character set: '%s'

- Error: 1116 SQLSTATE: HY000 (ER_TOO_MANY_TABLES)

Message: Too many tables; MySQL can only use %d tables in a join

- Error: 1117 SQLSTATE: HY000 (ER_TOO_MANY_FIELDS)

Message: Too many columns

- Error: 1118 SQLSTATE: 42000 (ER_TOO_BIG_ROW_SIZE)

Message: Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some columns to TEXT or BLOBs

- Error: 1119 SQLSTATE: HY000 (ER_STACK_OVERRUN)

Message: Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld -O thread_stack=#' to specify a bigger stack if needed

- Error: 1120 SQLSTATE: 42000 (ER_WRONG_OUTER_JOIN)

Message: Cross dependency found in OUTER JOIN; examine your ON conditions

- Error: 1121 SQLSTATE: 42000 (ER_NULL_COLUMN_IN_INDEX)

Message: Column '%s' is used with UNIQUE or INDEX but is not defined as NOT NULL

- Error: 1122 SQLSTATE: HY000 (ER_CANT_FIND_UDF)

Message: Can't load function '%s'

- Error: 1123 SQLSTATE: HY000 (ER_CANT_INITIALIZE_UDF)

Message: Can't initialize function '%s'; %s

- Error: 1124 SQLSTATE: HY000 (ER_UDF_NO_PATHS)

Message: No paths allowed for shared library

- Error: 1125 SQLSTATE: HY000 (ER_UDF_EXISTS)
Message: Function '%s' already exists
- Error: 1126 SQLSTATE: HY000 (ER_CANT_OPEN_LIBRARY)
Message: Can't open shared library '%s' (errno: %d %s)
- Error: 1127 SQLSTATE: HY000 (ER_CANT_FIND_DL_ENTRY)
Message: Can't find function '%s' in library
- Error: 1128 SQLSTATE: HY000 (ER_FUNCTION_NOT_DEFINED)
Message: Function '%s' is not defined
- Error: 1129 SQLSTATE: HY000 (ER_HOST_IS_BLOCKED)
Message: Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
- Error: 1130 SQLSTATE: HY000 (ER_HOST_NOT_PRIVILEGED)
Message: Host '%s' is not allowed to connect to this MySQL server
- Error: 1131 SQLSTATE: 42000 (ER_PASSWORD_ANONYMOUS_USER)
Message: You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords
- Error: 1132 SQLSTATE: 42000 (ER_PASSWORD_NOT_ALLOWED)
Message: You must have privileges to update tables in the mysql database to be able to change passwords for others
- Error: 1133 SQLSTATE: 42000 (ER_PASSWORD_NO_MATCH)
Message: Can't find any matching row in the user table
- Error: 1134 SQLSTATE: HY000 (ER_UPDATE_INFO)

Message: Rows matched: %ld Changed: %ld Warnings: %ld

- Error: 1135 SQLSTATE: HY000 (ER_CANT_CREATE_THREAD)

Message: Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug

- Error: 1136 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT_ON_ROW)

Message: Column count doesn't match value count at row %ld

- Error: 1137 SQLSTATE: HY000 (ER_CANT_REOPEN_TABLE)

Message: Can't reopen table: '%s'

- Error: 1138 SQLSTATE: 22004 (ER_INVALID_USE_OF_NULL)

Message: Invalid use of NULL value

- Error: 1139 SQLSTATE: 42000 (ER_REGEXP_ERROR)

Message: Got error '%s' from regexp

- Error: 1140 SQLSTATE: 42000 (ER_MIX_OF_GROUP_FUNC_AND_FIELDS)

Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

- Error: 1141 SQLSTATE: 42000 (ER_NONEXISTING_GRANT)

Message: There is no such grant defined for user '%s' on host '%s'

- Error: 1142 SQLSTATE: 42000 (ER_TABLEACCESS_DENIED_ERROR)

Message: %s command denied to user '%s'@'%s' for table '%s'

- Error: 1143 SQLSTATE: 42000 (ER_COLUMNACCESS_DENIED_ERROR)

Message: %s command denied to user '%s'@'%s' for column '%s' in table '%s'

- Error: 1144 SQLSTATE: 42000 (ER_ILLEGAL_GRANT_FOR_TABLE)

Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used

- Error: 1145 SQLSTATE: 42000 (ER_GRANT_WRONG_HOST_OR_USER)

Message: The host or user argument to GRANT is too long

- Error: 1146 SQLSTATE: 42S02 (ER_NO_SUCH_TABLE)

Message: Table '%s.%s' doesn't exist

- Error: 1147 SQLSTATE: 42000 (ER_NONEXISTING_TABLE_GRANT)

Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error: 1148 SQLSTATE: 42000 (ER_NOT_ALLOWED_COMMAND)

Message: The used command is not allowed with this MySQL version

- Error: 1149 SQLSTATE: 42000 (ER_SYNTAX_ERROR)

Message: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use

- Error: 1150 SQLSTATE: HY000 (ER_DELAYED_CANT_CHANGE_LOCK)

Message: Delayed insert thread couldn't get requested lock for table %s

- Error: 1151 SQLSTATE: HY000 (ER_TOO_MANY_DELAYED_THREADS)

Message: Too many delayed threads in use

- Error: 1152 SQLSTATE: 08S01 (ER_ABORTING_CONNECTION)

Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error: 1153 SQLSTATE: 08S01 (ER_NET_PACKET_TOO_LARGE)

Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error: 1154 SQLSTATE: 08S01 (ER_NET_READ_ERROR_FROM_PIPE)

Message: Got a read error from the connection pipe

- Error: 1155 SQLSTATE: 08S01 (ER_NET_FCNTL_ERROR)

Message: Got an error from fcntl()

- Error: 1156 SQLSTATE: 08S01 (ER_NET_PACKETS_OUT_OF_ORDER)

Message: Got packets out of order

- Error: 1157 SQLSTATE: 08S01 (ER_NET_UNCOMPRESS_ERROR)

Message: Couldn't uncompress communication packet

- Error: 1158 SQLSTATE: 08S01 (ER_NET_READ_ERROR)

Message: Got an error reading communication packets

- Error: 1159 SQLSTATE: 08S01 (ER_NET_READ_INTERRUPTED)

Message: Got timeout reading communication packets

- Error: 1160 SQLSTATE: 08S01 (ER_NET_ERROR_ON_WRITE)

Message: Got an error writing communication packets

- Error: 1161 SQLSTATE: 08S01 (ER_NET_WRITE_INTERRUPTED)

Message: Got timeout writing communication packets

- Error: 1162 SQLSTATE: 42000 (ER_TOO_LONG_STRING)

Message: Result string is longer than 'max_allowed_packet' bytes

- Error: 1163 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_BLOB)

Message: The used table type doesn't support BLOB/TEXT columns

- Error: 1164 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_AUTO_INCREMENT)

Message: The used table type doesn't support AUTO_INCREMENT columns

- Error: 1165 SQLSTATE: HY000 (ER_DELAYED_INSERT_TABLE_LOCKED)

Message: INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES

- Error: 1166 SQLSTATE: 42000 (ER_WRONG_COLUMN_NAME)

Message: Incorrect column name '%s'

- Error: 1167 SQLSTATE: 42000 (ER_WRONG_KEY_COLUMN)

Message: The used storage engine can't index column '%s'

- Error: 1168 SQLSTATE: HY000 (ER_WRONG_MRG_TABLE)

Message: All tables in the MERGE table are not identically defined

- Error: 1169 SQLSTATE: 23000 (ER_DUP_UNIQUE)

Message: Can't write, because of unique constraint, to table '%s'

- Error: 1170 SQLSTATE: 42000 (ER_BLOB_KEY_WITHOUT_LENGTH)

Message: BLOB/TEXT column '%s' used in key specification without a key length

- Error: 1171 SQLSTATE: 42000 (ER_PRIMARY_CANT_HAVE_NULL)

Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead

- Error: 1172 SQLSTATE: 42000 (ER_TOO_MANY_ROWS)

Message: Result consisted of more than one row

- Error: 1173 SQLSTATE: 42000 (ER_REQUIRES_PRIMARY_KEY)

Message: This table type requires a primary key

- Error: 1174 SQLSTATE: HY000 (ER_NO_RAID_COMPILED)

Message: This version of MySQL is not compiled with RAID support

- Error: 1175 SQLSTATE: HY000 (ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE)

Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column

- Error: 1176 SQLSTATE: HY000 (ER_KEY_DOES_NOT_EXISTS)

Message: Key '%s' doesn't exist in table '%s'

- Error: 1177 SQLSTATE: 42000 (ER_CHECK_NO_SUCH_TABLE)

Message: Can't open table

- Error: 1178 SQLSTATE: 42000 (ER_CHECK_NOT_IMPLEMENTED)

Message: The storage engine for the table doesn't support %s

- Error: 1179 SQLSTATE: 25000
(ER_CANT_DO_THIS_DURING_AN_TRANSACTION)

Message: You are not allowed to execute this command in a transaction

- Error: 1180 SQLSTATE: HY000 (ER_ERROR_DURING_COMMIT)

Message: Got error %d during COMMIT

- Error: 1181 SQLSTATE: HY000 (ER_ERROR_DURING_ROLLBACK)

Message: Got error %d during ROLLBACK

- Error: 1182 SQLSTATE: HY000 (ER_ERROR_DURING_FLUSH_LOGS)

Message: Got error %d during FLUSH_LOGS

- Error: 1183 SQLSTATE: HY000 (ER_ERROR_DURING_CHECKPOINT)

Message: Got error %d during CHECKPOINT

- Error: 1184 SQLSTATE: 08S01 (ER_NEW_ABORTING_CONNECTION)

Message: Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s)

- Error: 1185 SQLSTATE: HY000 (ER_DUMP_NOT_IMPLEMENTED)

Message: The storage engine for the table does not support binary table dump

- Error: 1186 SQLSTATE: HY000 (ER_FLUSH_MASTER_BINLOG_CLOSED)

Message: Binlog closed, cannot RESET MASTER

- Error: 1187 SQLSTATE: HY000 (ER_INDEX_REBUILD)

Message: Failed rebuilding the index of dumped table '%s'

- Error: 1188 SQLSTATE: HY000 (ER_MASTER)

Message: Error from master: '%s'

- Error: 1189 SQLSTATE: 08S01 (ER_MASTER_NET_READ)

Message: Net error reading from master

- Error: 1190 SQLSTATE: 08S01 (ER_MASTER_NET_WRITE)

Message: Net error writing to master

- Error: 1191 SQLSTATE: HY000 (ER_FT_MATCHING_KEY_NOT_FOUND)

Message: Can't find FULLTEXT index matching the column list

- Error: 1192 SQLSTATE: HY000 (ER_LOCK_OR_ACTIVE_TRANSACTION)

Message: Can't execute the given command because you have active locked tables or an active transaction

- Error: 1193 SQLSTATE: HY000 (ER_UNKNOWN_SYSTEM_VARIABLE)

Message: Unknown system variable '%s'

- Error: 1194 SQLSTATE: HY000 (ER_CRASHED_ON_USAGE)

Message: Table '%s' is marked as crashed and should be repaired

- Error: 1195 SQLSTATE: HY000 (ER_CRASHED_ON_REPAIR)

Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error: 1196 SQLSTATE: HY000 (ER_WARNING_NOT_COMPLETE_ROLLBACK)

Message: Some non-transactional changed tables couldn't be rolled back

- Error: 1197 SQLSTATE: HY000 (ER_TRANS_CACHE_FULL)

Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again

- Error: 1198 SQLSTATE: HY000 (ER_SLAVE_MUST_STOP)

Message: This operation cannot be performed with a running slave; run STOP SLAVE first

- Error: 1199 SQLSTATE: HY000 (ER_SLAVE_NOT_RUNNING)

Message: This operation requires a running slave; configure slave and do START SLAVE

- Error: 1200 SQLSTATE: HY000 (ER_BAD_SLAVE)

Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO

- Error: 1201 SQLSTATE: HY000 (ER_MASTER_INFO)

Message: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error: 1202 SQLSTATE: HY000 (ER_SLAVE_THREAD)

Message: Could not create slave thread; check system resources

- Error: 1203 SQLSTATE: 42000 (ER_TOO_MANY_USER_CONNECTIONS)

Message: User %s already has more than 'max_user_connections' active connections

- Error: 1204 SQLSTATE: HY000 (ER_SET_CONSTANTS_ONLY)

Message: You may only use constant expressions with SET

- Error: 1205 SQLSTATE: HY000 (ER_LOCK_WAIT_TIMEOUT)

Message: Lock wait timeout exceeded; try restarting transaction

- Error: 1206 SQLSTATE: HY000 (ER_LOCK_TABLE_FULL)

Message: The total number of locks exceeds the lock table size

- Error: 1207 SQLSTATE: 25000 (ER_READ_ONLY_TRANSACTION)

Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error: 1208 SQLSTATE: HY000 (ER_DROP_DB_WITH_READ_LOCK)

Message: DROP DATABASE not allowed while thread is holding global read lock

- Error: 1209 SQLSTATE: HY000 (ER_CREATE_DB_WITH_READ_LOCK)

Message: CREATE DATABASE not allowed while thread is holding global read lock

- Error: 1210 SQLSTATE: HY000 (ER_WRONG_ARGUMENTS)

Message: Incorrect arguments to %s

- Error: 1211 SQLSTATE: 42000 (ER_NO_PERMISSION_TO_CREATE_USER)

Message: '%s'@'%s' is not allowed to create new users

- Error: 1212 SQLSTATE: HY000 (ER_UNION_TABLES_IN_DIFFERENT_DIR)

Message: Incorrect table definition; all MERGE tables must be in the same database

- Error: 1213 SQLSTATE: 40001 (ER_LOCK_DEADLOCK)

Message: Deadlock found when trying to get lock; try restarting transaction

- Error: 1214 SQLSTATE: HY000 (ER_TABLE_CANT_HANDLE_FT)

Message: The used table type doesn't support FULLTEXT indexes

- Error: 1215 SQLSTATE: HY000 (ER_CANNOT_ADD_FOREIGN)

Message: Cannot add foreign key constraint

- Error: 1216 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW)

Message: Cannot add or update a child row: a foreign key constraint fails

- Error: 1217 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED)

Message: Cannot delete or update a parent row: a foreign key constraint fails

- Error: 1218 SQLSTATE: 08S01 (ER_CONNECT_TO_MASTER)

Message: Error connecting to master: %s

- Error: 1219 SQLSTATE: HY000 (ER_QUERY_ON_MASTER)

Message: Error running query on master: %s

- Error: 1220 SQLSTATE: HY000 (ER_ERROR_WHEN_EXECUTING_COMMAND)

Message: Error when executing command %s: %s

- Error: 1221 SQLSTATE: HY000 (ER_WRONG_USAGE)

Message: Incorrect usage of %s and %s

- Error: 1222 SQLSTATE: 21000
(ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT)

Message: The used SELECT statements have a different number of columns

- Error: 1223 SQLSTATE: HY000 (ER_CANT_UPDATE_WITH_READLOCK)

Message: Can't execute the query because you have a conflicting read lock

- Error: 1224 SQLSTATE: HY000 (ER_MIXING_NOT_ALLOWED)

Message: Mixing of transactional and non-transactional tables is disabled

- Error: 1225 SQLSTATE: HY000 (ER_DUP_ARGUMENT)

Message: Option '%s' used twice in statement

- Error: 1226 SQLSTATE: 42000 (ER_USER_LIMIT_REACHED)

Message: User '%s' has exceeded the '%s' resource (current value: %ld)

- Error: 1227 SQLSTATE: 42000 (ER_SPECIFIC_ACCESS_DENIED_ERROR)

Message: Access denied; you need the %s privilege for this operation

- Error: 1228 SQLSTATE: HY000 (ER_LOCAL_VARIABLE)

Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL

- Error: 1229 SQLSTATE: HY000 (ER_GLOBAL_VARIABLE)

Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL

- Error: 1230 SQLSTATE: 42000 (ER_NO_DEFAULT)

Message: Variable '%s' doesn't have a default value

- Error: 1231 SQLSTATE: 42000 (ER_WRONG_VALUE_FOR_VAR)

Message: Variable '%s' can't be set to the value of '%s'

- Error: 1232 SQLSTATE: 42000 (ER_WRONG_TYPE_FOR_VAR)

Message: Incorrect argument type to variable '%s'

- Error: 1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ)

Message: Variable '%s' can only be set, not read

- Error: 1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE)

Message: Incorrect usage/placement of '%s'

- Error: 1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET)

Message: This version of MySQL doesn't yet support '%s'

- Error: 1236 SQLSTATE: HY000
(ER_MASTER_FATAL_ERROR_READING_BINLOG)

Message: Got fatal error %d: '%s' from master when reading data from binary log

- Error: 1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE)

Message: Slave SQL thread ignored the query because of replicate-*-table rules

- Error: 1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR)

Message: Variable '%s' is a %s variable

- Error: 1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF)

Message: Incorrect foreign key definition for '%s': %s

- Error: 1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF)

Message: Key reference and table reference don't match

- Error: 1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS)
Message: Operand should contain %d column(s)
- Error: 1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW)
Message: Subquery returns more than 1 row
- Error: 1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER)
Message: Unknown prepared statement handler (%.*s) given to %s
- Error: 1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB)
Message: Help database is corrupt or does not exist
- Error: 1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE)
Message: Cyclic reference on subqueries
- Error: 1246 SQLSTATE: HY000 (ER_AUTO_CONVERT)
Message: Converting column '%s' from %s to %s
- Error: 1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE)
Message: Reference '%s' not supported (%s)
- Error: 1248 SQLSTATE: 42000 (ER_DERIVED_MUST_HAVE_ALIAS)
Message: Every derived table must have its own alias
- Error: 1249 SQLSTATE: 01000 (ER_SELECT_REDUCED)
Message: Select %u was reduced during optimization
- Error: 1250 SQLSTATE: 42000 (ER_TABLENAME_NOT_ALLOWED_HERE)
Message: Table '%s' from one of the SELECTs cannot be used in %s
- Error: 1251 SQLSTATE: 08004 (ER_NOT_SUPPORTED_AUTH_MODE)

Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client

- Error: 1252 SQLSTATE: 42000 (ER_SPATIAL_CANT_HAVE_NULL)

Message: All parts of a SPATIAL index must be NOT NULL

- Error: 1253 SQLSTATE: 42000 (ER_COLLATION_CHARSET_MISMATCH)

Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

- Error: 1254 SQLSTATE: HY000 (ER_SLAVE_WAS_RUNNING)

Message: Slave is already running

- Error: 1255 SQLSTATE: HY000 (ER_SLAVE_WAS_NOT_RUNNING)

Message: Slave already has been stopped

- Error: 1256 SQLSTATE: HY000 (ER_TOO_BIG_FOR_UNCOMPRESS)

Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)

- Error: 1257 SQLSTATE: HY000 (ER_ZLIB_Z_MEM_ERROR)

Message: ZLIB: Not enough memory

- Error: 1258 SQLSTATE: HY000 (ER_ZLIB_Z_BUF_ERROR)

Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error: 1259 SQLSTATE: HY000 (ER_ZLIB_Z_DATA_ERROR)

Message: ZLIB: Input data corrupted

- Error: 1260 SQLSTATE: HY000 (ER_CUT_VALUE_GROUP_CONCAT)

Message: %d line(s) were cut by GROUP_CONCAT()

- Error: 1261 SQLSTATE: 01000 (ER_WARN_TOO_FEW_RECORDS)
Message: Row %ld doesn't contain data for all columns
- Error: 1262 SQLSTATE: 01000 (ER_WARN_TOO_MANY_RECORDS)
Message: Row %ld was truncated; it contained more data than there were input columns
- Error: 1263 SQLSTATE: 22004 (ER_WARN_NULL_TO_NOTNULL)
Message: Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld
- Error: 1264 SQLSTATE: 22003 (ER_WARN_DATA_OUT_OF_RANGE)
Message: Out of range value adjusted for column '%s' at row %ld
- Error: 1265 SQLSTATE: 01000 (WARN_DATA_TRUNCATED)
Message: Data truncated for column '%s' at row %ld
- Error: 1266 SQLSTATE: HY000 (ER_WARN_USING_OTHER_HANDLER)
Message: Using storage engine %s for table '%s'
- Error: 1267 SQLSTATE: HY000 (ER_CANT_AGGREGATE_2COLLATIONS)
Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'
- Error: 1268 SQLSTATE: HY000 (ER_DROP_USER)
Message: Cannot drop one or more of the requested users
- Error: 1269 SQLSTATE: HY000 (ER_REVOKE_GRANTS)
Message: Can't revoke all privileges for one or more of the requested users
- Error: 1270 SQLSTATE: HY000 (ER_CANT_AGGREGATE_3COLLATIONS)
Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation

'%s'

- Error: 1271 SQLSTATE: HY000 (ER_CANT_AGGREGATE_NCOLLATIONS)

Message: Illegal mix of collations for operation '%s'

- Error: 1272 SQLSTATE: HY000 (ER_VARIABLE_IS_NOT_STRUCT)

Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error: 1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION)

Message: Unknown collation: '%s'

- Error: 1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS)

Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE)

Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format

- Error: 1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED)

Message: Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d

- Error: 1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND)

Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL

- Error: 1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE)

Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart

- Error: 1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED)
Message: SQL thread is not to be started so UNTIL options are ignored
- Error: 1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX)
Message: Incorrect index name '%s'
- Error: 1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG)
Message: Incorrect catalog name '%s'
- Error: 1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE)
Message: Query cache failed to set size %lu; new query cache size is %lu
- Error: 1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN)
Message: Column '%s' cannot be part of FULLTEXT index
- Error: 1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE)
Message: Unknown key cache '%s'
- Error: 1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK)
Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work
- Error: 1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE)
Message: Unknown table engine '%s'
- Error: 1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX)
Message: '%s' is deprecated; use '%s' instead
- Error: 1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE)
Message: The target table %s of the %s is not updatable

- Error: 1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED)

Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working

- Error: 1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT)

Message: The MySQL server is running with the %s option so it cannot execute this statement

- Error: 1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE)

Message: Column '%s' has duplicated value '%s' in %s

- Error: 1292 SQLSTATE: 22007 (ER_TRUNCATED_WRONG_VALUE)

Message: Truncated incorrect %s value: '%s'

- Error: 1293 SQLSTATE: HY000 (ER_TOO_MUCH_AUTO_TIMESTAMP_COLS)

Message: Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

- Error: 1294 SQLSTATE: HY000 (ER_INVALID_ON_UPDATE)

Message: Invalid ON UPDATE clause for '%s' column

- Error: 1295 SQLSTATE: HY000 (ER_UNSUPPORTED_PS)

Message: This command is not supported in the prepared statement protocol yet

- Error: 1296 SQLSTATE: HY000 (ER_GET_ERRMSG)

Message: Got error %d '%s' from %s

- Error: 1297 SQLSTATE: HY000 (ER_GET_TEMPORARY_ERRMSG)

Message: Got temporary error %d '%s' from %s

- Error: 1298 SQLSTATE: HY000 (ER_UNKNOWN_TIME_ZONE)
Message: Unknown or incorrect time zone: '%s'
- Error: 1299 SQLSTATE: HY000 (ER_WARN_INVALID_TIMESTAMP)
Message: Invalid TIMESTAMP value in column '%s' at row %ld
- Error: 1300 SQLSTATE: HY000 (ER_INVALID_CHARACTER_STRING)
Message: Invalid %s character string: '%s'
- Error: 1301 SQLSTATE: HY000 (ER_WARN_ALLOWED_PACKET_OVERFLOWED)
Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated
- Error: 1302 SQLSTATE: HY000 (ER_CONFLICTING_DECLARATIONS)
Message: Conflicting declarations: '%s%s' and '%s%s'
- Error: 1303 SQLSTATE: 2F003 (ER_SP_NO_RECURSIVE_CREATE)
Message: Can't create a %s from within another stored routine
- Error: 1304 SQLSTATE: 42000 (ER_SP_ALREADY_EXISTS)
Message: %s %s already exists
- Error: 1305 SQLSTATE: 42000 (ER_SP_DOES_NOT_EXIST)
Message: %s %s does not exist
- Error: 1306 SQLSTATE: HY000 (ER_SP_DROP_FAILED)
Message: Failed to DROP %s %s
- Error: 1307 SQLSTATE: HY000 (ER_SP_STORE_FAILED)
Message: Failed to CREATE %s %s

- Error: 1308 SQLSTATE: 42000 (ER_SP_LILABEL_MISMATCH)
Message: %s with no matching label: %s
- Error: 1309 SQLSTATE: 42000 (ER_SP_LABEL_REDEFINE)
Message: Redefining label %s
- Error: 1310 SQLSTATE: 42000 (ER_SP_LABEL_MISMATCH)
Message: End-label %s without match
- Error: 1311 SQLSTATE: 01000 (ER_SP_UNINIT_VAR)
Message: Referring to uninitialized variable %s
- Error: 1312 SQLSTATE: 0A000 (ER_SP_BADSELECT)
Message: PROCEDURE %s can't return a result set in the given context
- Error: 1313 SQLSTATE: 42000 (ER_SP_BADRETURN)
Message: RETURN is only allowed in a FUNCTION
- Error: 1314 SQLSTATE: 0A000 (ER_SP_BADSTATEMENT)
Message: %s is not allowed in stored procedures
- Error: 1315 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_IGNORED)
Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored
- Error: 1316 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_TRANSLATED)
Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN
- Error: 1317 SQLSTATE: 70100 (ER_QUERY_INTERRUPTED)
Message: Query execution was interrupted

- Error: 1318 SQLSTATE: 42000 (ER_SP_WRONG_NO_OF_ARGS)
Message: Incorrect number of arguments for %s %s; expected %u, got %u
- Error: 1319 SQLSTATE: 42000 (ER_SP_COND_MISMATCH)
Message: Undefined CONDITION: %s
- Error: 1320 SQLSTATE: 42000 (ER_SP_NORETURN)
Message: No RETURN found in FUNCTION %s
- Error: 1321 SQLSTATE: 2F005 (ER_SP_NORETURNEND)
Message: FUNCTION %s ended without RETURN
- Error: 1322 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_QUERY)
Message: Cursor statement must be a SELECT
- Error: 1323 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_SELECT)
Message: Cursor SELECT must not have INTO
- Error: 1324 SQLSTATE: 42000 (ER_SP_CURSOR_MISMATCH)
Message: Undefined CURSOR: %s
- Error: 1325 SQLSTATE: 24000 (ER_SP_CURSOR_ALREADY_OPEN)
Message: Cursor is already open
- Error: 1326 SQLSTATE: 24000 (ER_SP_CURSOR_NOT_OPEN)
Message: Cursor is not open
- Error: 1327 SQLSTATE: 42000 (ER_SP_UNDECLARED_VAR)
Message: Undeclared variable: %s
- Error: 1328 SQLSTATE: HY000 (ER_SP_WRONG_NO_OF_FETCH_ARGS)

Message: Incorrect number of FETCH variables

- Error: 1329 SQLSTATE: 02000 (ER_SP_FETCH_NO_DATA)

Message: No data - zero rows fetched, selected, or processed

- Error: 1330 SQLSTATE: 42000 (ER_SP_DUP_PARAM)

Message: Duplicate parameter: %s

- Error: 1331 SQLSTATE: 42000 (ER_SP_DUP_VAR)

Message: Duplicate variable: %s

- Error: 1332 SQLSTATE: 42000 (ER_SP_DUP_COND)

Message: Duplicate condition: %s

- Error: 1333 SQLSTATE: 42000 (ER_SP_DUP_CURS)

Message: Duplicate cursor: %s

- Error: 1334 SQLSTATE: HY000 (ER_SP_CANT_ALTER)

Message: Failed to ALTER %s %s

- Error: 1335 SQLSTATE: 0A000 (ER_SP_SUBSELECT_NYI)

Message: Subselect value not supported

- Error: 1336 SQLSTATE: 0A000 (ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG)

Message: %s is not allowed in stored function or trigger

- Error: 1337 SQLSTATE: 42000 (ER_SP_VARCOND_AFTER_CURSHNDLR)

Message: Variable or condition declaration after cursor or handler declaration

- Error: 1338 SQLSTATE: 42000 (ER_SP_CURSOR_AFTER_HANDLER)

Message: Cursor declaration after handler declaration

- Error: 1339 SQLSTATE: 20000 (ER_SP_CASE_NOT_FOUND)

Message: Case not found for CASE statement

- Error: 1340 SQLSTATE: HY000 (ER_FPARSER_TOO_BIG_FILE)

Message: Configuration file '%s' is too big

- Error: 1341 SQLSTATE: HY000 (ER_FPARSER_BAD_HEADER)

Message: Malformed file type header in file '%s'

- Error: 1342 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_COMMENT)

Message: Unexpected end of file while parsing comment '%s'

- Error: 1343 SQLSTATE: HY000 (ER_FPARSER_ERROR_IN_PARAMETER)

Message: Error while parsing parameter '%s' (line: '%s')

- Error: 1344 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER)

Message: Unexpected end of file while skipping unknown parameter '%s'

- Error: 1345 SQLSTATE: HY000 (ER_VIEW_NO_EXPLAIN)

Message: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table

- Error: 1346 SQLSTATE: HY000 (ER_FRM_UNKNOWN_TYPE)

Message: File '%s' has unknown type '%s' in its header

- Error: 1347 SQLSTATE: HY000 (ER_WRONG_OBJECT)

Message: '%s.%s' is not %s

- Error: 1348 SQLSTATE: HY000 (ER_NONUPDATEABLE_COLUMN)

Message: Column '%s' is not updatable

- Error: 1349 SQLSTATE: HY000 (ER_VIEW_SELECT_DERIVED)

Message: View's SELECT contains a subquery in the FROM clause

- Error: 1350 SQLSTATE: HY000 (ER_VIEW_SELECT_CLAUSE)

Message: View's SELECT contains a '%s' clause

- Error: 1351 SQLSTATE: HY000 (ER_VIEW_SELECT_VARIABLE)

Message: View's SELECT contains a variable or parameter

- Error: 1352 SQLSTATE: HY000 (ER_VIEW_SELECT_TMPTABLE)

Message: View's SELECT refers to a temporary table '%s'

- Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)

Message: View's SELECT and view's field list have different column counts

- Error: 1354 SQLSTATE: HY000 (ER_WARN_VIEW_MERGE)

Message: View merge algorithm can't be used here for now (assumed undefined algorithm)

- Error: 1355 SQLSTATE: HY000 (ER_WARN_VIEW_WITHOUT_KEY)

Message: View being updated does not have complete key of underlying table in it

- Error: 1356 SQLSTATE: HY000 (ER_VIEW_INVALID)

Message: View '%s.%s' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them

- Error: 1357 SQLSTATE: HY000 (ER_SP_NO_DROP_SP)

Message: Can't drop or alter a %s from within another stored routine

- Error: 1358 SQLSTATE: HY000 (ER_SP_GOTO_IN_HNDLR)
Message: GOTO is not allowed in a stored procedure handler
- Error: 1359 SQLSTATE: HY000 (ER_TRG_ALREADY_EXISTS)
Message: Trigger already exists
- Error: 1360 SQLSTATE: HY000 (ER_TRG_DOES_NOT_EXIST)
Message: Trigger does not exist
- Error: 1361 SQLSTATE: HY000 (ER_TRG_ON_VIEW_OR_TEMP_TABLE)
Message: Trigger's '%s' is view or temporary table
- Error: 1362 SQLSTATE: HY000 (ER_TRG_CANT_CHANGE_ROW)
Message: Updating of %s row is not allowed in %strigger
- Error: 1363 SQLSTATE: HY000 (ER_TRG_NO_SUCH_ROW_IN_TRG)
Message: There is no %s row in %s trigger
- Error: 1364 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_FIELD)
Message: Field '%s' doesn't have a default value
- Error: 1365 SQLSTATE: 22012 (ER_DIVISION_BY_ZERO)
Message: Division by 0
- Error: 1366 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE_FOR_FIELD)
Message: Incorrect %s value: '%s' for column '%s' at row %ld
- Error: 1367 SQLSTATE: 22007 (ER_ILLEGAL_VALUE_FOR_TYPE)
Message: Illegal %s '%s' value found during parsing
- Error: 1368 SQLSTATE: HY000 (ER_VIEW_NONUPD_CHECK)

Message: CHECK OPTION on non-updatable view '%s.%s'

- Error: 1369 SQLSTATE: HY000 (ER_VIEW_CHECK_FAILED)

Message: CHECK OPTION failed '%s.%s'

- Error: 1370 SQLSTATE: 42000 (ER_PROCAccess_DENIED_ERROR)

Message: %s command denied to user '%s'@'%s' for routine '%s'

- Error: 1371 SQLSTATE: HY000 (ER_RELAY_LOG_FAIL)

Message: Failed purging old relay logs: %s

- Error: 1372 SQLSTATE: HY000 (ER_PASSWD_LENGTH)

Message: Password hash should be a %d-digit hexadecimal number

- Error: 1373 SQLSTATE: HY000 (ER_UNKNOWN_TARGET_BINLOG)

Message: Target log not found in binlog index

- Error: 1374 SQLSTATE: HY000 (ER_IO_ERR_LOG_INDEX_READ)

Message: I/O error reading log index file

- Error: 1375 SQLSTATE: HY000 (ER_BINLOG_PURGE_PROHIBITED)

Message: Server configuration does not permit binlog purge

- Error: 1376 SQLSTATE: HY000 (ER_FSEEK_FAIL)

Message: Failed on fseek()

- Error: 1377 SQLSTATE: HY000 (ER_BINLOG_PURGE_FATAL_ERR)

Message: Fatal error during log purge

- Error: 1378 SQLSTATE: HY000 (ER_LOG_IN_USE)

Message: A purgeable log is in use, will not purge

- Error: 1379 SQLSTATE: HY000 (ER_LOG_PURGE_UNKNOWN_ERR)
Message: Unknown error during log purge
- Error: 1380 SQLSTATE: HY000 (ER_RELAY_LOG_INIT)
Message: Failed initializing relay log position: %s
- Error: 1381 SQLSTATE: HY000 (ER_NO_BINARY_LOGGING)
Message: You are not using binary logging
- Error: 1382 SQLSTATE: HY000 (ER_RESERVED_SYNTAX)
Message: The '%s' syntax is reserved for purposes internal to the MySQL server
- Error: 1383 SQLSTATE: HY000 (ER_WSAS_FAILED)
Message: WSAStartup Failed
- Error: 1384 SQLSTATE: HY000 (ER_DIFF_GROUPS_PROC)
Message: Can't handle procedures with different groups yet
- Error: 1385 SQLSTATE: HY000 (ER_NO_GROUP_FOR_PROC)
Message: Select must have a group with this procedure
- Error: 1386 SQLSTATE: HY000 (ER_ORDER_WITH_PROC)
Message: Can't use ORDER clause with this procedure
- Error: 1387 SQLSTATE: HY000 (ER_LOGGING_PROHIBIT_CHANGING_OF)
Message: Binary logging and replication forbid changing the global server %s
- Error: 1388 SQLSTATE: HY000 (ER_NO_FILE_MAPPING)
Message: Can't map file: %s, errno: %d

- Error: 1389 SQLSTATE: HY000 (ER_WRONG_MAGIC)
Message: Wrong magic in %s
- Error: 1390 SQLSTATE: HY000 (ER_PS_MANY_PARAM)
Message: Prepared statement contains too many placeholders
- Error: 1391 SQLSTATE: HY000 (ER_KEY_PART_0)
Message: Key part '%s' length cannot be 0
- Error: 1392 SQLSTATE: HY000 (ER_VIEW_CHECKSUM)
Message: View text checksum failed
- Error: 1393 SQLSTATE: HY000 (ER_VIEW_MULTIUPDATE)
Message: Can not modify more than one base table through a join view '%s.%s'
- Error: 1394 SQLSTATE: HY000 (ER_VIEW_NO_INSERT_FIELD_LIST)
Message: Can not insert into join view '%s.%s' without fields list
- Error: 1395 SQLSTATE: HY000 (ER_VIEW_DELETE_MERGE_VIEW)
Message: Can not delete from join view '%s.%s'
- Error: 1396 SQLSTATE: HY000 (ER_CANNOT_USER)
Message: Operation %s failed for %s
- Error: 1397 SQLSTATE: XAE04 (ER_XAER_NOTA)
Message: XAER_NOTA: Unknown XID
- Error: 1398 SQLSTATE: XAE05 (ER_XAER_INVAL)
Message: XAER_INVAL: Invalid arguments (or unsupported command)

- Error: 1399 SQLSTATE: XAE07 (ER_XAER_RMFAIL)

Message: XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state

- Error: 1400 SQLSTATE: XAE09 (ER_XAER_OUTSIDE)

Message: XAER_OUTSIDE: Some work is done outside global transaction

- Error: 1401 SQLSTATE: XAE03 (ER_XAER_RMERR)

Message: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency

- Error: 1402 SQLSTATE: XA100 (ER_XA_RBROLLBACK)

Message: XA_RBROLLBACK: Transaction branch was rolled back

- Error: 1403 SQLSTATE: 42000 (ER_NONEXISTING_PROC_GRANT)

Message: There is no such grant defined for user '%s' on host '%s' on routine '%s'

- Error: 1404 SQLSTATE: HY000 (ER_PROC_AUTO_GRANT_FAIL)

Message: Failed to grant EXECUTE and ALTER ROUTINE privileges

- Error: 1405 SQLSTATE: HY000 (ER_PROC_AUTO_REVOKE_FAIL)

Message: Failed to revoke all privileges to dropped routine

- Error: 1406 SQLSTATE: 22001 (ER_DATA_TOO_LONG)

Message: Data too long for column '%s' at row %ld

- Error: 1407 SQLSTATE: 42000 (ER_SP_BAD_SQLSTATE)

Message: Bad SQLSTATE: '%s'

- Error: 1408 SQLSTATE: HY000 (ER_STARTUP)

Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s

- Error: 1409 SQLSTATE: HY000 (ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR)

Message: Can't load value from file with fixed size rows to variable

- Error: 1410 SQLSTATE: 42000 (ER_CANT_CREATE_USER_WITH_GRANT)

Message: You are not allowed to create a user with GRANT

- Error: 1411 SQLSTATE: HY000 (ER_WRONG_VALUE_FOR_TYPE)

Message: Incorrect %s value: '%s' for function %s

- Error: 1412 SQLSTATE: HY000 (ER_TABLE_DEF_CHANGED)

Message: Table definition has changed, please retry transaction

- Error: 1413 SQLSTATE: 42000 (ER_SP_DUP_HANDLER)

Message: Duplicate handler declared in the same block

- Error: 1414 SQLSTATE: 42000 (ER_SP_NOT_VAR_ARG)

Message: OUT or INOUT argument %d for routine %s is not a variable or NEW pseudo-variable in BEFORE trigger

- Error: 1415 SQLSTATE: 0A000 (ER_SP_NO_RESET)

Message: Not allowed to return a result set from a %s

- Error: 1416 SQLSTATE: 22003 (ER_CANT_CREATE_GEOMETRY_OBJECT)

Message: Cannot get geometry object from data you send to the GEOMETRY field

- Error: 1417 SQLSTATE: HY000 (ER_FAILED_ROUTINE_BREAK_BINLOG)

Message: A routine failed and has neither NO SQL nor READS SQL DATA in its declaration and binary logging is enabled; if non-transactional tables were updated, the binary log will miss their changes

- Error: 1418 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_ROUTINE)

Message: This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

- Error: 1419 SQLSTATE: HY000 (ER_BINLOG_CREATE_ROUTINE_NEED_SUPER)

Message: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

- Error: 1420 SQLSTATE: HY000 (ER_EXEC_STMT_WITH_OPEN_CURSOR)

Message: You can't execute a prepared statement which has an open cursor associated with it. Reset the statement to re-execute it.

- Error: 1421 SQLSTATE: HY000 (ER_STMT_HAS_NO_OPEN_CURSOR)

Message: The statement (%lu) has no open cursor.

- Error: 1422 SQLSTATE: HY000 (ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG)

Message: Explicit or implicit commit is not allowed in stored function or trigger.

- Error: 1423 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_VIEW_FIELD)

Message: Field of view '%s.%s' underlying table doesn't have a default value

- Error: 1424 SQLSTATE: HY000 (ER_SP_NO_RECURSION)

Message: Recursive stored functions and triggers are not allowed.

- Error: 1425 SQLSTATE: 42000 (ER_TOO_BIG_SCALE)

Message: Too big scale %d specified for column '%s'. Maximum is %d.

- Error: 1426 SQLSTATE: 42000 (ER_TOO_BIG_PRECISION)

Message: Too big precision %d specified for column '%s'. Maximum is %d.

- Error: 1427 SQLSTATE: 42000 (ER_M_BIGGER_THAN_D)

Message: For float(M,D), double(M,D) or decimal(M,D), M must be >= D (column '%s').

- Error: 1428 SQLSTATE: HY000 (ER_WRONG_LOCK_OF_SYSTEM_TABLE)

Message: You can't combine write-locking of system '%s.%s' table with other tables

- Error: 1429 SQLSTATE: HY000 (ER_CONNECT_TO_FOREIGN_DATA_SOURCE)

Message: Unable to connect to foreign data source: %s

- Error: 1430 SQLSTATE: HY000 (ER_QUERY_ON_FOREIGN_DATA_SOURCE)

Message: There was a problem processing the query on the foreign data source. Data source error: %-.64

- Error: 1431 SQLSTATE: HY000 (ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST)

Message: The foreign data source you are trying to reference does not exist. Data source error: %s

- Error: 1432 SQLSTATE: HY000
(ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE)

Message: Can't create federated table. The data source connection string '%s' is not in the correct format

- Error: 1433 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID)

Message: The data source connection string '%s' is not in the correct format

- Error: 1434 SQLSTATE: HY000 (ER_CANT_CREATE_FEDERATED_TABLE)

Message: Can't create federated table. Foreign data src error: %s

- Error: 1435 SQLSTATE: HY000 (ER_TRG_IN_WRONG_SCHEMA)

Message: Trigger in wrong schema

- Error: 1436 SQLSTATE: HY000 (ER_STACK_OVERRUN_NEED_MORE)

Message: Thread stack overrun: %ld bytes used of a %ld byte stack, and %ld bytes needed. Use 'mysqld -O thread_stack=#' to specify a bigger stack.

- Error: 1437 SQLSTATE: 42000 (ER_TOO_LONG_BODY)

Message: Routine body for '%s' is too long

- Error: 1438 SQLSTATE: HY000 (ER_WARN_CANT_DROP_DEFAULT_KEYCACHE)

Message: Cannot drop default keycache

- Error: 1439 SQLSTATE: 42000 (ER_TOO_BIG_DISPLAYWIDTH)

Message: Display width out of range for column '%s' (max = %d)

- Error: 1440 SQLSTATE: XAE08 (ER_XAER_DUPID)

Message: XAER_DUPID: The XID already exists

- Error: 1441 SQLSTATE: 22008 (ER_DATETIME_FUNCTION_OVERFLOW)

Message: Datetime function: %s field overflow

- Error: 1442 SQLSTATE: HY000
(ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG)

Message: Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.

- Error: 1443 SQLSTATE: HY000 (ER_VIEW_PREVENT_UPDATE)

Message: The definition of table '%s' prevents operation %s on table '%s'.

- Error: 1444 SQLSTATE: HY000 (ER_PS_NO_RECURSION)

Message: The prepared statement contains a stored routine call that refers to

that same statement. It's not allowed to execute a prepared statement in such a recursive manner

- Error: 1445 SQLSTATE: HY000 (ER_SP_CANT_SET_AUTOCOMMIT)

Message: Not allowed to set autocommit from a stored function or trigger

- Error: 1446 SQLSTATE: HY000 (ER_MALFORMED_DEFINER)

Message: Definer is not fully qualified

- Error: 1447 SQLSTATE: HY000 (ER_VIEW_FRM_NO_USER)

Message: View '%s'.'%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!

- Error: 1448 SQLSTATE: HY000 (ER_VIEW_OTHER_USER)

Message: You need the SUPER privilege for creation view with '%s'@'%s' definer

- Error: 1449 SQLSTATE: HY000 (ER_NO_SUCH_USER)

Message: There is no '%s'@'%s' registered

- Error: 1450 SQLSTATE: HY000 (ER_FORBID_SCHEMA_CHANGE)

Message: Changing schema from '%s' to '%s' is not allowed.

- Error: 1451 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED_2)

Message: Cannot delete or update a parent row: a foreign key constraint fails (%s)

- Error: 1452 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW_2)

Message: Cannot add or update a child row: a foreign key constraint fails (%s)

- Error: 1453 SQLSTATE: 42000 (ER_SP_BAD_VAR_SHADOW)

Message: Variable '%s' must be quoted with `...`, or renamed

- Error: 1454 SQLSTATE: HY000 (ER_TRG_NO_DEFINER)

Message: No definer attribute for trigger '%s'.%s'. The trigger will be activated under the authorization of the invoker, which may have insufficient privileges. Please recreate the trigger.

- Error: 1455 SQLSTATE: HY000 (ER_OLD_FILE_FORMAT)

Message: '%s' has an old format, you should re-create the '%s' object(s)

- Error: 1456 SQLSTATE: HY000 (ER_SP_RECURSION_LIMIT)

Message: Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s

- Error: 1457 SQLSTATE: HY000 (ER_SP_PROC_TABLE_CORRUPT)

Message: Failed to load routine %s. The table mysql.proc is missing, corrupt, or contains bad data (internal code %d)

- Error: 1458 SQLSTATE: 42000 (ER_SP_WRONG_NAME)

Message: Incorrect routine name '%s'

- Error: 1459 SQLSTATE: HY000 (ER_TABLE_NEEDS_UPGRADE)

Message: Table upgrade required. Please do "REPAIR TABLE `'%s`" to fix it!

- Error: 1460 SQLSTATE: 42000 (ER_SP_NO_AGGREGATE)

Message: AGGREGATE is not supported for stored functions

- Error: 1461 SQLSTATE: 42000 (ER_MAX_PREPARED_STMT_COUNT_REACHED)

Message: Can't create more than max_prepared_stmt_count statements (current value: %lu)

- Error: 1462 SQLSTATE: HY000 (ER_VIEW_RECURSIVE)

Message: ``%s`.`%s` contains view recursion

- Error: 1463 SQLSTATE: 42000 (ER_NON_GROUPING_FIELD_USED)

Message: non-grouping field '%s' is used in %s clause

- Error: 1464 SQLSTATE: HY000 (ER_TABLE_CANT_HANDLE_SPKEYS)

Message: The used table type doesn't support SPATIAL indexes

- Error: 1465 SQLSTATE: HY000 (ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA)

Message: Triggers can not be created on system tables

- Error: 1466 SQLSTATE: HY000 (ER_REMOVED_SPACES)

Message: Leading spaces are removed from name '%s'

B.2. Client Error Codes and Messages

Client error information comes from the following source files:

- The Error values and the symbols in parentheses correspond to definitions in the `include/errmsg.h` MySQL source file.
- The Message values correspond to the error messages that are listed in the `libmysql/errmsg.c` file. `%d` and `%s` represent numbers and strings, respectively, that are substituted into the messages when they are displayed.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: 2000 (CR_UNKNOWN_ERROR)
Message: Unknown MySQL error
- Error: 2001 (CR_SOCKET_CREATE_ERROR)
Message: Can't create UNIX socket (%d)
- Error: 2002 (CR_CONNECTION_ERROR)
Message: Can't connect to local MySQL server through socket '%s' (%d)
- Error: 2003 (CR_CONN_HOST_ERROR)
Message: Can't connect to MySQL server on '%s' (%d)
- Error: 2004 (CR_IPSOCK_ERROR)
Message: Can't create TCP/IP socket (%d)
- Error: 2005 (CR_UNKNOWN_HOST)
Message: Unknown MySQL server host '%s' (%d)
- Error: 2006 (CR_SERVER_GONE_ERROR)

Message: MySQL server has gone away

- Error: 2007 (CR_VERSION_ERROR)

Message: Protocol mismatch; server version = %d, client version = %d

- Error: 2008 (CR_OUT_OF_MEMORY)

Message: MySQL client ran out of memory

- Error: 2009 (CR_WRONG_HOST_INFO)

Message: Wrong host info

- Error: 2010 (CR_LOCALHOST_CONNECTION)

Message: Localhost via UNIX socket

- Error: 2011 (CR_TCP_CONNECTION)

Message: %s via TCP/IP

- Error: 2012 (CR_SERVER_HANDSHAKE_ERR)

Message: Error in server handshake

- Error: 2013 (CR_SERVER_LOST)

Message: Lost connection to MySQL server during query

- Error: 2014 (CR_COMMANDS_OUT_OF_SYNC)

Message: Commands out of sync; you can't run this command now

- Error: 2015 (CR_NAMEDPIPE_CONNECTION)

Message: Named pipe: %s

- Error: 2016 (CR_NAMEDPIPEWAIT_ERROR)

Message: Can't wait for named pipe to host: %s pipe: %s (%lu)

- Error: 2017 (CR_NAMEDPIPEOPEN_ERROR)

Message: Can't open named pipe to host: %s pipe: %s (%lu)

- Error: 2018 (CR_NAMEDPIPESETSTATE_ERROR)

Message: Can't set state of named pipe to host: %s pipe: %s (%lu)

- Error: 2019 (CR_CANT_READ_CHARSET)

Message: Can't initialize character set %s (path: %s)

- Error: 2020 (CR_NET_PACKET_TOO_LARGE)

Message: Got packet bigger than 'max_allowed_packet' bytes

- Error: 2021 (CR_EMBEDDED_CONNECTION)

Message: Embedded server

- Error: 2022 (CR_PROBE_SLAVE_STATUS)

Message: Error on SHOW SLAVE STATUS:

- Error: 2023 (CR_PROBE_SLAVE_HOSTS)

Message: Error on SHOW SLAVE HOSTS:

- Error: 2024 (CR_PROBE_SLAVE_CONNECT)

Message: Error connecting to slave:

- Error: 2025 (CR_PROBE_MASTER_CONNECT)

Message: Error connecting to master:

- Error: 2026 (CR_SSL_CONNECTION_ERROR)

Message: SSL connection error

- Error: 2027 (CR_MALFORMED_PACKET)

Message: Malformed packet

- Error: 2028 (CR_WRONG_LICENSE)

Message: This client library is licensed only for use with MySQL servers having '%s' license

- Error: 2029 (CR_NULL_POINTER)

Message: Invalid use of null pointer

- Error: 2030 (CR_NO_PREPARE_STMT)

Message: Statement not prepared

- Error: 2031 (CR_PARAMS_NOT_BOUND)

Message: No data supplied for parameters in prepared statement

- Error: 2032 (CR_DATA_TRUNCATED)

Message: Data truncated

- Error: 2033 (CR_NO_PARAMETERS_EXISTS)

Message: No parameters exist in the statement

- Error: 2034 (CR_INVALID_PARAMETER_NO)

Message: Invalid parameter number

- Error: 2035 (CR_INVALID_BUFFER_USE)

Message: Can't send long data for non-string/non-binary data types (parameter: %d)

- Error: 2036 (CR_UNSUPPORTED_PARAM_TYPE)

Message: Using unsupported buffer type: %d (parameter: %d)

- Error: 2037 (CR_SHARED_MEMORY_CONNECTION)

Message: Shared memory: %s

- Error: 2038 (CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR)

Message: Can't open shared memory; client could not create request event (%lu)

- Error: 2039 (CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR)

Message: Can't open shared memory; no answer event received from server (%lu)

- Error: 2040 (CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR)

Message: Can't open shared memory; server could not allocate file mapping (%lu)

- Error: 2041 (CR_SHARED_MEMORY_CONNECT_MAP_ERROR)

Message: Can't open shared memory; server could not get pointer to file mapping (%lu)

- Error: 2042 (CR_SHARED_MEMORY_FILE_MAP_ERROR)

Message: Can't open shared memory; client could not allocate file mapping (%lu)

- Error: 2043 (CR_SHARED_MEMORY_MAP_ERROR)

Message: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error: 2044 (CR_SHARED_MEMORY_EVENT_ERROR)

Message: Can't open shared memory; client could not create %s event (%lu)

- Error: 2045 (CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR)

Message: Can't open shared memory; no answer from server (%lu)

- Error: 2046 (CR_SHARED_MEMORY_CONNECT_SET_ERROR)

Message: Can't open shared memory; cannot send request event to server (%lu)

- Error: 2047 (CR_CONN_UNKNOW_PROTOCOL)

Message: Wrong or unknown protocol

- Error: 2048 (CR_INVALID_CONN_HANDLE)

Message: Invalid connection handle

- Error: 2049 (CR_SECURE_AUTH)

Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)

- Error: 2050 (CR_FETCH_CANCELED)

Message: Row retrieval was canceled by mysql_stmt_close() call

- Error: 2051 (CR_NO_DATA)

Message: Attempt to read column without prior row fetch

- Error: 2052 (CR_NO_STMT_METADATA)

Message: Prepared statement contains no metadata

- Error: 2053 (CR_NO_RESULT_SET)

Message: Attempt to read a row while there is no result set associated with the statement

- Error: 2054 (CR_NOT_IMPLEMENTED)

Message: This feature is not implemented yet

Appendix C. Credits

Table of Contents

[C.1. Developers at MySQL AB](#)

[C.2. Contributors to MySQL](#)

[C.3. Documenters and translators](#)

[C.4. Libraries used by and included with MySQL](#)

[C.5. Packages that support MySQL](#)

[C.6. Tools that were used to create MySQL](#)

[C.7. Supporters of MySQL](#)

This appendix lists the developers, contributors, and supporters that have helped to make MySQL what it is today.

C.1. Developers at MySQL AB

These are the developers that are or have been employed by MySQL AB to work on the MySQL database software, roughly in the order they started to work with us. Following each developer is a small list of the tasks that the developer is responsible for, or the accomplishments they have made. All developers are involved in support.

- Michael (Monty) Widenius
 - Lead developer and main author of the MySQL server (**mysqld**).
 - New functions for the string library.
 - Most of the `mysys` library.
 - The `ISAM` and `MyISAM` libraries (B-tree index file handlers with index compression and different record formats).
 - The `HEAP` library. A memory table system with our superior full dynamic hashing. In use since 1981 and published around 1984.
 - The **replace** program (take a look at it, it's **COOL!**).
 - Connector/ODBC (MyODBC), the ODBC driver for Windows.
 - Fixing bugs in MIT-pthreads to get it to work for MySQL Server. And also Unireg, a curses-based application tool with many utilities.
 - Porting of `mSQL` tools like `msqlperl`, `DBD/DBI`, and `DB2mysql`.
 - Most of `crash-me` and the foundation for the MySQL benchmarks.
- David Axmark
 - Initial main writer of the **Reference Manual**, including enhancements to **texi2html**.
 - Automatic Web site updating from the manual.

- Initial Autoconf, Automake, and Libtool support.
- Licensing.
- Parts of all the text files. (Nowadays only the README is left. The rest ended up in the manual.)
- Lots of testing of new features.
- Our in-house Free Software legal expert.
- Mailing list maintainer (who never has the time to do it right...).
- Our original portability code (now more than 10 years old). Nowadays only some parts of mysys are left.
- Someone for Monty to call in the middle of the night when he just got that new feature to work.
- Chief "Open Sourcerer" (MySQL community relations).
- Jani Tolonen
 - **mysqlimport**
 - A lot of extensions to the command-line clients.
 - PROCEDURE ANALYSE()
- Sinisa Milivojevic (now in support)
 - Compression (with zlib) in the client/server protocol.
 - Perfect hashing for the lexical analyzer phase.
 - Multi-row INSERT
 - **mysqldump** -e option
 - LOAD DATA LOCAL INFILE

- SQL_CALC_FOUND_ROWS SELECT option
- --max-user-connections=... option
- net_read and net_write_timeout
- GRANT/REVOKE and SHOW GRANTS FOR
- New client/server protocol for 4.0
- UNION in 4.0
- Multiple-table DELETE/UPDATE
- Subqueries in the FROM clause (4.1).
- User resources management
- Initial developer of the MySQL++ C++ API and the MySQLGUI client.
- Tonu Samuel (past developer)
 - VIO interface (the foundation for the encrypted client/server protocol).
 - MySQL Filesystem (a way to use MySQL databases as files and directories).
 - The CASE expression.
 - The MD5() and COALESCE() functions.
 - RAID support for MyISAM tables.
- Sasha Pachev (past developer)
 - Initial implementation of replication (up to version 4.0).
 - SHOW CREATE TABLE.
 - mysql-bench

- Matt Wagner
 - MySQL test suite.
 - Webmaster (until 2002).
- Miguel Solorzano (now in support)
 - Win32 development and release builds.
 - Windows NT server code.
 - WinMySQLAdmin
- Timothy Smith (now in support)
 - Dynamic character sets support.
 - configure, RPMs and other parts of the build system.
 - Initial developer of `libmysqld`, the embedded server.
- Sergei Golubchik
 - Full-text search.
 - Added keys to the MERGE library.
 - Precision math.
- Jeremy Cole (past developer)
 - Proofreading and editing this fine manual.
 - `ALTER TABLE ... ORDER BY`
 - `UPDATE ... ORDER BY`
 - `DELETE ... ORDER BY`
- Indrek Siitan

- Designing/programming of our Web interface.
- Author of our newsletter management system.
- Jorge del Conde (now in support)
 - **MySQLCC (MySQL Control Center)**
 - Win32 development
 - Initial implementation of the Web site portals.
- Venu Anuganti (past developer)
 - MyODBC 3.51
 - New client/server protocol for 4.1 (for prepared statements).
- Arjen Lentz (now handling community)
 - Maintainer of the MySQL Reference Manual.
 - Preparing the O'Reilly printed edition of the manual.
- Alexander (Bar) Barkov, Alexey (Holyfoot) Botchkov, and Ramil Kalimullin
 - Spatial data (GIS) and R-Trees implementation for 4.1
 - Unicode and character sets for 4.1; documentation for same
- Oleksandr (Sanja) Byelkin
 - Query cache in 4.0
 - Implementation of subqueries (4.1).
 - Implementation of views (5.0).
- Aleksey (Walrus) Kishkin and Alexey (Ranger) Stroganov

- Benchmarks design and analysis.
- Maintenance of the MySQL test suite.
- Zak Greant (past employee)
 - Open Source advocate, MySQL community relations.
- Carsten Pedersen
 - The MySQL Certification program.
- Lenz Grimmer
 - Production (build and release) engineering.
- Peter Zaitsev
 - SHA1(), AES_ENCRYPT() and AES_DECRYPT() functions.
 - Debugging, cleaning up various features.
- Alexander (Salle) Keremidarski
 - Support.
 - Debugging.
- Per-Erik Martin
 - Lead developer for stored procedures (5.0).
- Jim Winstead
 - Former lead Web developer.
 - Improving server, fixing bugs.
- Mark Matthews
 - Connector/J driver (Java).

- Peter Gultzan
 - SQL standards compliance.
 - Documentation of existing MySQL code/algorithms.
 - Character set documentation.
- Guilhem Bichot
 - Replication, from MySQL version 4.0.
 - Fixed handling of exponents for DECIMAL.
 - Author of `mysql_tableinfo`.
 - Backup (in 5.1).
- Antony T. Curtis
 - Porting of the MySQL Database software to OS/2.
- Mikael Ronstrom
 - Much of the initial work on NDB Cluster until 2000. Roughly half the code base at that time. Transaction protocol, node recovery, system restart and restart code and parts of the API functionality.
 - Lead Architect, developer, debugger of NDB Cluster 1994-2004
 - Lots of optimizations
- Jonas Orelund
 - On-line Backup
 - The automatic test environment of MySQL Cluster
 - Portability Library for NDB Cluster
 - Lots of other things

- Pekka Nouisiainen
 - Ordered index implementation of MySQL Cluster
 - BLOB support in MySQL Cluster
 - Charset support in MySQL Cluster
- Martin Skold
 - Unique index implementation of MySQL Cluster
 - Integration of NDB Cluster into MySQL
- Magnus Svensson
 - The test framework for MySQL Cluster
 - Integration of NDB Cluster into MySQL
- Tomas Ulin
 - Lots of work on configuration changes for simple installation and use of MySQL Cluster
- Konstantin Osipov
 - Prepared statements.
 - Cursors.
- Dmitri Lenev
 - Time zone support.
 - Triggers (in 5.0).

C.2. Contributors to MySQL

Although MySQL AB owns all copyrights in the MySQL server and the MySQL manual, we wish to recognize those who have made contributions of one kind or another to the MySQL distribution. Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola <gwerq@mbx.vol.it> or <gwerq@tin.it>

The initial port to Win32/NT.

- Per Eric Olsson

For more or less constructive criticism and real testing of the dynamic record format.

- Irena Pancirov <irena@mail.yacc.it>

Win32 port with Borland compiler. `mysqlshutdown.exe` and `mysqlwatch.exe`

- David J. Hughes

For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with mSQL, but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. **mysqladmin** and **mysql** client are programs that were largely influenced by their mSQL counterparts. We have put a lot of effort into making the MySQL syntax a superset of mSQL. Many of the API's ideas are borrowed from mSQL to make it easy to port free mSQL programs to the MySQL API. The MySQL software doesn't contain any code from mSQL. Two files in the distribution (`client/insert_test.c` and `client/select_test.c`) are based on the corresponding (non-copyrighted) files in the mSQL distribution, but are modified as examples showing the changes necessary to convert code from mSQL to MySQL Server. (mSQL is copyrighted David J. Hughes.)

- Patrick Lynch

For helping us acquire <http://www.mysql.com/>.

- Fred Lindberg

For setting up gmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko <igor@frog.kiev.ua>

mysqldump (previously msqldump, but ported and enhanced by Monty).

- Yuri Dario

For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

Author of **mysqlhotcopy**.

- Zarko Mocnik <zarko.mocnik@dem.si>

Sorting for Slovenian language.

- "TAMITO" <tommy@valley.ne.jp>

The `_MB` character set macros and the `ujis` and `sjis` character sets.

- Joshua Chamas <joshua@chamas.com>

Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier <Yves.Carlier@rug.ac.be>

mysqlaccess, a program to show the access rights for a user.

- Rhys Jones <rhys@wales.com> (And GWE Technologies Limited)

For one of the early JDBC drivers.

- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>

Further development of one of the early JDBC drivers and other MySQL-related Java tools.

- James Cooper <pixel@organic.com>

For setting up a searchable mailing list archive at his site.

- Rick Mehalick <Rick_Mehalick@i-o.com>

For xmysql, a graphical X client for MySQL Server.

- Doug Sisk <sisk@wix.com>

For providing RPM packages of MySQL for Red Hat Linux.

- Diemand Alexander V. <axeld@vial.ethz.ch>

For providing RPM packages of MySQL for Red Hat Linux-Alpha.

- Antoni Pamies Olive <toni@readysoft.es>

For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

- Jay Bloodworth <jay@pathways.sde.state.sc.us>

For providing RPM versions for MySQL 3.21.

- David Sacerdote <davids@secnet.com>

Ideas for secure checking of DNS hostnames.

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

Some support for Chinese(BIG5) characters.

- Wei He <hewei@mail.ied.ac.cn>

A lot of functionality for the Chinese(GBK) character set.

- Jan Pazdziora <adelton@fi.muni.cz>

Czech sorting order.

- Zeev Suraski <bourbon@netvision.net.il>

FROM_UNIXTIME() time formatting, ENCRYPT() functions, and **bison** advisor.
Active mailing list member.

- Luuk de Boer <luuk@wxs.nl>

Ported (and extended) the benchmark suite to DBI/DBD. Have been of great help with crash-me and running benchmarks. Some new date functions.
The **mysql_setpermission** script.

- Alexis Mikhailov <root@medinf.chuvashia.su>

User-defined functions (UDFs); CREATE FUNCTION and DROP FUNCTION.

- Andreas F. Bobak <bobak@relog.ch>

The AGGREGATE extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

The libmysql.dll library.

- James Pereria <jpereira@iafrica.com>

Mysqlmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>

Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.

- Martin Ramsch <m.ramsch@computer.org>

Examples in the MySQL Tutorial.

- Steve Harvey

For making **mysqlaccess** more secure.

- Konark IA-64 Centre of Persistent Systems Private Limited

<http://www.pspl.co.in/konark/>. Help with the Win64 port of the MySQL server.

- Albert Chin-A-Young.

Configure updates for Tru64, large file support and better TCP wrappers support.

- John Birrell

Emulation of pthread_mutex() for OS/2.

- Benjamin Pflugmann

Extended MERGE tables to handle INSERTS. Active member on the MySQL mailing lists.

- Jocelyn Fournier

Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).

- Marc Liyanage

Maintaining the Mac OS X packages and providing invaluable feedback on how to create Mac OS X PKGs.

- Robert Rutherford

Providing invaluable information and feedback about the QNX port.

- Previous developers of NDB Cluster

Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to

mention here. Notable name is Ataullah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architectural foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, <jehamby@lightside>, <psmith@BayNetworks.com>, <duane@connect.com.au>, Ted Deppner <ted@psyber.com>, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch <dkoch@amcity.com>

Irix setup.

- Luuk de Boer <luuk@wxs.nl>

Benchmark questions.

- Tim Sailer <tps@users.buoy.com>

DBD::mysql questions.

- Boyd Lynn Gerber <gerberb@zenez.com>

SCO-related questions.

- Richard Mehalick <RM186061@shellus.com>

xmysql-related questions and basic installation questions.

- Zeev Suraski <bourbon@netvision.net.il>

Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

- Francesc Guasch <frankie@citel.upc.es>

General questions.

- Jonathan J Smith <jsmith@wtp.net>

Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar <sklar@student.net>

Using MySQL from PHP and Perl.

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

Is flexible and can handle Linux and perhaps HP-UX. Tries to get users to use **mysqlbug**.

- John Lyon <jlyon@imag.net>

Questions about installing MySQL on Linux systems, using either .rpm files or compiling from source.

- Lorvid Ltd. <lorvid@WOLFENET.com>

Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

DBD, Linux, some SQL syntax questions.

C.3. Documenters and translators

The following people have helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded (<http://mysql.hitstar.com/>) versions were based. [Personal home page at linuxdb.yeah.net](http://linuxdb.yeah.net).

- Jay Flaherty <fty@mediapulse.com>

Big parts of the Perl DBI/DBD section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaise.fr>

French error messages.

- Petr Snajdr, <snajdr@pvt.net>
Czech error messages.
- Jaroslaw Lewandowski <jotel@itnet.com.pl>
Polish error messages.
- Miguel Angel Fernandez Roiz
Spanish error messages.
- Roy-Magne Mo <rmo@www.hivolda.no>
Norwegian error messages and testing of MySQL 3.21.xx.
- Timur I. Bakeyev <root@timur.tatarstan.ru>
Russian error messages.
- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>
Italian error messages.
- Dirk Munzinger <dirk@trinity.saar.de>
German error messages.
- Billik Stefan <billik@sun.uniag.sk>
Slovak error messages.
- Stefan Saroiu <tzoompy@cs.washington.edu>
Romanian error messages.
- Peter Feher
Hungarian error messages.
- Roberto M. Serqueira

Portuguese error messages.

- Carsten H. Pedersen

Danish error messages.

- Arjen G. Lentz

Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

C.4. Libraries used by and included with MySQL

The following is a list of the creators of the libraries we have included with the MySQL server source to make it easy to compile and install MySQL. We are very thankful to all individuals that have created these and it has made our life much easier.

- Fred Fish

For his excellent C debugging and trace library. Monty has made a number of smaller improvements to the library (speed and additional options).

- Richard A. O'Keefe

For his public domain string library.

- Henry Spencer

For his regex library, used in `WHERE column REGEXP regexp`.

- Chris Provenzano

Portable user level pthreads. From the copyright: This product includes software developed by Chris Provenzano, the University of California, Berkeley, and contributors. We are currently using version 1_60_beta6 patched by Monty (see `mit-pthreads/Changes-mysql`).

- Jean-loup Gailly and Mark Adler

For the zlib library (used on MySQL on Windows).

- Bjorn Benson

For his `safe_malloc` (memory checker) package which is used in when you configure MySQL with `--debug`.

- Free Software Foundation

The `readline` library (used by the **mysql** command-line client).

- The NetBSD foundation

The `libedit` package (optionally used by the **mysql** command-line client).

C.5. Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We can't list every possible package here because the list would then be way too hard to maintain. For other packages, please refer to the software portal at <http://solutions.mysql.com/software/>.

- Tim Bunce, Alligator Descartes

For the DBD (Perl) interface.

- Andreas Koenig <a.koenig@mind.de>

For the Perl interface for MySQL Server.

- Jochen Wiedmann <wiedmann@neckar-alb.de>

For maintaining the Perl DBD::mysql module.

- Eugene Chan <eugene@acenet.com.sg>

For porting PHP for MySQL Server.

- Georg Richter

MySQL 4.1 testing and bug hunting. New PHP 5.0 mysql_i extension (API) for use with MySQL 4.1 and up.

- Giovanni Maruzzelli <maruzz@matrice.it>

For porting iODBC (Unix ODBC).

- Xavier Leroy <Xavier.Leroy@inria.fr>

The author of LinuxThreads (used by the MySQL Server on Linux).

C.6. Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation

From whom we got an excellent compiler (**gcc**), an excellent debugger (**gdb**) and the `libc` library (from which we have borrowed `strtoc` to get some code working in Linux).

- Free Software Foundation & The XEmacs development team

For a really great editor/environment used by almost everybody at MySQL AB.

- Julian Seward

Author of `valgrind`, an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.

- Dorothea Lütkehaus and Andreas Zeller

For `DDD` (The Data Display Debugger) which is an excellent graphical front end to **gdb**.

C.7. Supporters of MySQL

Although MySQL AB owns all copyrights in the MySQL server and the MySQL manual, we wish to recognize the following companies, which helped us finance the development of the MySQL server, such as by paying us for developing a new feature or giving us hardware for development of the MySQL server.

- VA Linux / Andover.net

Funded replication.

- NuSphere

Editing of the MySQL manual.

- Stork Design studio

The MySQL Web site in use between 1998-2000.

- Intel

Contributed to development on Windows and Linux platforms.

- Compaq

Contributed to Development on Linux/Alpha.

- SWSoft

Development on the embedded **mysqld** version.

- FutureQuest

--skip-show-database

Appendix D. MySQL Change History

Table of Contents

[D.1. Changes in release 5.0.x \(Production\)](#)

[D.1.1. Changes in release 5.0.25 \(Not yet released\)](#)

[D.1.2. Changes in release 5.0.24 \(Not yet released\)](#)

[D.1.3. Changes in release 5.0.23 \(Not released\)](#)

[D.1.4. Changes in release 5.0.22 \(24 May 2006\)](#)

[D.1.5. Changes in release 5.0.21 \(02 May 2006\)](#)

[D.1.6. Changes in release 5.0.20a \(18 April 2006\)](#)

[D.1.7. Changes in release 5.0.20 \(31 March 2006\)](#)

[D.1.8. Changes in release 5.0.19 \(04 March 2006\)](#)

[D.1.9. Changes in release 5.0.18 \(21 December 2005\)](#)

[D.1.10. Changes in release 5.0.17 \(14 December 2005\)](#)

[D.1.11. Changes in release 5.0.16 \(10 November 2005\)](#)

[D.1.12. Changes in release 5.0.15 \(19 October 2005: Production\)](#)

[D.1.13. Changes in release 5.0.14 \(Not released\)](#)

[D.1.14. Changes in release 5.0.13 \(22 September 2005: Release Candidate\)](#)

[D.1.15. Changes in release 5.0.12 \(02 September 2005\)](#)

[D.1.16. Changes in release 5.0.11 \(06 August 2005\)](#)

[D.1.17. Changes in release 5.0.10 \(27 July 2005\)](#)

[D.1.18. Changes in release 5.0.9 \(15 July 2005\)](#)

[D.1.19. Changes in release 5.0.8 \(Not released\)](#)

[D.1.20. Changes in release 5.0.7 \(10 June 2005\)](#)

[D.1.21. Changes in release 5.0.6 \(26 May 2005\)](#)

[D.1.22. Changes in release 5.0.5 \(Not released\)](#)

[D.1.23. Changes in release 5.0.4 \(16 April 2005\)](#)

[D.1.24. Changes in release 5.0.3 \(23 March 2005: Beta\)](#)

[D.1.25. Changes in release 5.0.2 \(01 December 2004\)](#)

[D.1.26. Changes in release 5.0.1 \(27 July 2004\)](#)

[D.1.27. Changes in release 5.0.0 \(22 December 2003: Alpha\)](#)

[D.2. Changes in MySQL Cluster](#)

[D.2.1. Changes in MySQL Cluster-5.0.7 \(10 June 2005\)](#)

[D.2.2. Changes in MySQL Cluster-5.0.6 \(26 May 2005\)](#)

[D.2.3. Changes in MySQL Cluster-5.0.5 \(Not released\)](#)

[D.2.4. Changes in MySQL Cluster-5.0.4 \(16 April 2005\)](#)

- [D.2.5. Changes in MySQL Cluster-5.0.3 \(23 March 2005: Beta\)](#)
- [D.2.6. Changes in MySQL Cluster-5.0.1 \(27 July 2004\)](#)
- [D.2.7. Changes in MySQL Cluster-4.1.13 \(15 July 2005\)](#)
- [D.2.8. Changes in MySQL Cluster-4.1.12 \(13 May 2005\)](#)
- [D.2.9. Changes in MySQL Cluster-4.1.11 \(01 April 2005\)](#)
- [D.2.10. Changes in MySQL Cluster-4.1.10 \(12 February 2005\)](#)
- [D.2.11. Changes in MySQL Cluster-4.1.9 \(13 January 2005\)](#)
- [D.2.12. Changes in MySQL Cluster-4.1.8 \(14 December 2004\)](#)
- [D.2.13. Changes in MySQL Cluster-4.1.7 \(23 October 2004\)](#)
- [D.2.14. Changes in MySQL Cluster-4.1.6 \(10 October 2004\)](#)
- [D.2.15. Changes in MySQL Cluster-4.1.5 \(16 September 2004\)](#)
- [D.2.16. Changes in MySQL Cluster-4.1.4 \(31 August 2004\)](#)
- [D.2.17. Changes in MySQL Cluster-4.1.3 \(28 June 2004\)](#)
- [D.3. MySQL Connector/ODBC \(MyODBC\) Change History](#)
 - [D.3.1. Changes in MyODBC 3.51.13](#)
 - [D.3.2. Changes in MyODBC 3.51.12](#)
 - [D.3.3. Changes in MyODBC 3.51.11](#)
- [D.4. MySQL Connector/NET Change History](#)
 - [D.4.1. Version 1.0.8](#)
 - [D.4.2. Version 1.0.7](#)
 - [D.4.3. Version 1.0.6](#)
 - [D.4.4. Version 1.0.5](#)
 - [D.4.5. Version 1.0.4 1-20-05](#)
 - [D.4.6. Version 1.0.3-gamma 12-10-04](#)
 - [D.4.7. Version 1.0.2-gamma 04-11-15](#)
 - [D.4.8. Version 1.0.1-beta2 04-10-27](#)
 - [D.4.9. Version 1.0.0 04-09-01](#)
 - [D.4.10. Version 0.9.0 04-08-30](#)
 - [D.4.11. Version 0.76](#)
 - [D.4.12. Version 0.75](#)
 - [D.4.13. Version 0.74](#)
 - [D.4.14. Version 0.71](#)
 - [D.4.15. Version 0.70](#)
 - [D.4.16. Version 0.68](#)
 - [D.4.17. Version 0.65](#)
 - [D.4.18. Version 0.60](#)
 - [D.4.19. Version 0.50](#)
- [D.5. MySQL Connector/J Change History](#)
 - [D.5.1. Changes in MySQL Connector/J 5.0.2-beta \(11 July 2006\)](#)

[D.5.2. Changes in MySQL Connector/J 5.0.1-beta \(Not Released\)](#)
[D.5.3. Changes in MySQL Connector/J 5.0.0-beta \(22 December 2005\)](#)
[D.5.4. Changes in MySQL Connector/J 3.1.14 \(not yet released\)](#)
[D.5.5. Changes in MySQL Connector/J 3.1.13 \(26 May 2006\)](#)
[D.5.6. Changes in MySQL Connector/J 3.1.12 \(30 November 2005\)](#)
[D.5.7. Changes in MySQL Connector/J 3.1.11-stable \(07 October 2005\)](#)
[D.5.8. Changes in MySQL Connector/J 3.1.10-stable \(23 June 2005\)](#)
[D.5.9. Changes in MySQL Connector/J 3.1.9-stable \(22 June 2005\)](#)
[D.5.10. Changes in MySQL Connector/J 3.1.8-stable \(14 April 2005\)](#)
[D.5.11. Changes in MySQL Connector/J 3.1.7-stable \(18 February 2005\)](#)
[D.5.12. Changes in MySQL Connector/J 3.1.6-stable \(23 December 2004\)](#)
[D.5.13. Changes in MySQL Connector/J 3.1.5-gamma \(02 December 2004\)](#)
[D.5.14. Changes in MySQL Connector/J 3.1.4-beta \(04 September 2004\)](#)
[D.5.15. Changes in MySQL Connector/J 3.1.3-beta \(07 July 2004\)](#)
[D.5.16. Changes in MySQL Connector/J 3.1.2-alpha \(09 June 2004\)](#)
[D.5.17. Changes in MySQL Connector/J 3.1.1-alpha \(14 February 2004\)](#)
[D.5.18. Changes in MySQL Connector/J 3.1.0-alpha \(18 February 2003\)](#)
[D.5.19. Changes in MySQL Connector/J 3.0.17-ga \(23 June 2005\)](#)
[D.5.20. Changes in MySQL Connector/J 3.0.16-ga \(15 November 2004\)](#)
[D.5.21. Changes in MySQL Connector/J 3.0.15-production \(04 September 2004\)](#)
[D.5.22. Changes in MySQL Connector/J 3.0.14-production \(28 May 2004\)](#)
[D.5.23. Changes in MySQL Connector/J 3.0.13-production \(27 May 2004\)](#)
[D.5.24. Changes in MySQL Connector/J 3.0.12-production \(18 May 2004\)](#)
[D.5.25. Changes in MySQL Connector/J 3.0.11-stable \(19 February 2004\)](#)
[D.5.26. Changes in MySQL Connector/J 3.0.10-stable \(13 January 2004\)](#)
[D.5.27. Changes in MySQL Connector/J 3.0.9-stable \(07 October 2003\)](#)
[D.5.28. Changes in MySQL Connector/J 3.0.8-stable \(23 May 2003\)](#)
[D.5.29. Changes in MySQL Connector/J 3.0.7-stable \(08 April 2003\)](#)
[D.5.30. Changes in MySQL Connector/J 3.0.6-stable \(18 February 2003\)](#)
[D.5.31. Changes in MySQL Connector/J 3.0.5-gamma \(22 January 2003\)](#)
[D.5.32. Changes in MySQL Connector/J 3.0.4-gamma \(06 January 2003\)](#)
[D.5.33. Changes in MySQL Connector/J 3.0.3-dev \(17 December 2002\)](#)
[D.5.34. Changes in MySQL Connector/J 3.0.2-dev \(08 November 2002\)](#)
[D.5.35. Changes in MySQL Connector/J 3.0.1-dev \(21 September 2002\)](#)
[D.5.36. Changes in MySQL Connector/J 3.0.0-dev \(31 July 2002\)](#)
[D.5.37. Changes in MySQL Connector/J 2.0.14 \(16 May 2002\)](#)
[D.5.38. Changes in MySQL Connector/J 2.0.13 \(24 April 2002\)](#)
[D.5.39. Changes in MySQL Connector/J 2.0.12 \(07 April 2002\)](#)

- [D.5.40. Changes in MySQL Connector/J 2.0.11 \(27 January 2002\)](#)
- [D.5.41. Changes in MySQL Connector/J 2.0.10 \(24 January 2002\)](#)
- [D.5.42. Changes in MySQL Connector/J 2.0.9 \(13 January 2002\)](#)
- [D.5.43. Changes in MySQL Connector/J 2.0.8 \(25 November 2001\)](#)
- [D.5.44. Changes in MySQL Connector/J 2.0.7 \(24 October 2001\)](#)
- [D.5.45. Changes in MySQL Connector/J 2.0.6 \(16 June 2001\)](#)
- [D.5.46. Changes in MySQL Connector/J 2.0.5 \(13 June 2001\)](#)
- [D.5.47. Changes in MySQL Connector/J 2.0.3 \(03 December 2000\)](#)
- [D.5.48. Changes in MySQL Connector/J 2.0.1 \(06 April 2000\)](#)
- [D.5.49. Changes in MySQL Connector/J 2.0.0pre5 \(21 February 2000\)](#)
- [D.5.50. Changes in MySQL Connector/J 2.0.0pre4 \(10 January 2000\)](#)
- [D.5.51. Changes in MySQL Connector/J 2.0.0pre \(17 August 1999\)](#)
- [D.5.52. Changes in MySQL Connector/J 1.2b \(04 July 1999\)](#)
- [D.5.53. Changes in MySQL Connector/J 1.2a \(14 April 1999\)](#)
- [D.5.54. Changes in MySQL Connector/J 1.1i \(24 March 1999\)](#)
- [D.5.55. Changes in MySQL Connector/J 1.1h \(08 March 1999\)](#)
- [D.5.56. Changes in MySQL Connector/J 1.1g \(19 February 1999\)](#)
- [D.5.57. Changes in MySQL Connector/J 1.1f \(31 December 1998\)](#)
- [D.5.58. Changes in MySQL Connector/J 1.1b \(03 November 1998\)](#)
- [D.5.59. Changes in MySQL Connector/J 1.1 \(02 September 1998\)](#)
- [D.5.60. Changes in MySQL Connector/J 1.0 \(24 August 1998\)](#)
- [D.5.61. Changes in MySQL Connector/J 0.9d \(04 August 1998\)](#)
- [D.5.62. Changes in MySQL Connector/J 0.9 \(28 July 1998\)](#)
- [D.5.63. Changes in MySQL Connector/J 0.8 \(06 July 1998\)](#)
- [D.5.64. Changes in MySQL Connector/J 0.7 \(01 July 1998\)](#)
- [D.5.65. Changes in MySQL Connector/J 0.6 \(21 May 1998\)](#)

This appendix lists the changes from version to version in the MySQL source code through the latest version of MySQL 5.0, which is currently MySQL 5.0.25. Starting with MySQL 5.0, we began offering a new version of the Manual for each new series of MySQL releases (5.0, 5.1, and so on). For information about changes in previous release series of the MySQL database software, see the corresponding version of this Manual. For information about legacy versions of the MySQL software through the 4.1 series, see *MySQL 3.23, 4.0, 4.1 Reference Manual*.

We update this section as we add new features in the 5.0 series, so that everybody can follow the development process.

Note that we tend to update the manual at the same time we make changes to MySQL. If you find a recent version of MySQL listed here that you can't find on our download page (<http://dev.mysql.com/downloads/>), it means that the version has not yet been released.

The date mentioned with a release version is the date of the last BitKeeper ChangeSet on which the release was based, not the date when the packages were made available. The binaries are usually made available a few days after the date of the tagged ChangeSet, because building and testing all packages takes some time.

The manual included in the source and binary distributions may not be fully accurate when it comes to the release changelog entries, because the integration of the manual happens at build time. For the most up-to-date release changelog, please refer to the online version instead.

D.1. Changes in release 5.0.x (Production)

The following changelog shows what has been done in the 5.0 tree:

- Basic support for read-only server side cursors. For information about using cursors within stored routines, see [Section 17.2.9, “Cursors”](#). For information about using cursors from within the C API, see [Section 22.2.7.3, “mysql_stmt_attr_set\(\)”](#).
- Basic support for (updatable) views. See, for example, [Section 19.2, “CREATE VIEW Syntax”](#).
- Basic support for stored procedures and functions (SQL:2003 style). See [Chapter 17, *Stored Procedures and Functions*](#).
- Initial support for rudimentary triggers.
- Added `SELECT INTO list_of_vars`, which can be of mixed (that is, global and local) types. See [Section 17.2.7.3, “SELECT ... INTO Statement”](#).
- Removed the update log. It is fully replaced by the binary log. If the MySQL server is started with `--log-update`, it is translated to `--log-bin` (or ignored if the server is explicitly started with `--log-bin`), and a warning message is written to the error log. Setting `SQL_LOG_UPDATE` silently sets `SQL_LOG_BIN` instead (or do nothing if the server is explicitly started with `--log-bin`).
- Support for the ISAM storage engine has been removed. If you have ISAM tables, you should convert them before upgrading. See [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#).
- Support for RAID options in MyISAM tables has been removed. If you have tables that use these options, you should convert them before upgrading. See [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#).
- User variable names are now case insensitive: If you do `SET @a=10;` then `SELECT @A;` now returns 10. Case sensitivity of a variable's value depends on the collation of the value.

- Strict mode, which in essence means that you get an error instead of a warning when inserting an incorrect value into a column. See [Section 5.2.5, “The Server SQL Mode”](#).
- VARCHAR and VARBINARY columns remember end space. A VARCHAR() or VARBINARY column can contain up to 65,535 characters or bytes, respectively.
- MEMORY (HEAP) tables can have VARCHAR() columns.
- When using a constant string or a function that generates a string result in CREATE . . . SELECT, MySQL creates the result field based on the maximum length of the string or expression:

Maximum Length	Data type
= 0	CHAR(0)
< 512	VARCHAR(max_length)
>= 512	TEXT

For a full list of changes, please refer to the changelog sections for each individual 5.0.x release.

D.1.1. Changes in release 5.0.25 (Not yet released)

This is a bugfix release for the current production release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Network* (a commercial MySQL offering). For more details please see <http://www.mysql.com/network/advisors.html>.

Functionality added or changed:

- Table comments longer than 60 characters and column comments longer than 255 characters were truncated silently. Now a warning is issued, or an error in strict mode. (Bug #13934)

- The bundled yaSSL library was upgraded to version 1.3.7.
- The server now issues a warning if it removes leading spaces from an alias. (Bug #10977)

Bugs fixed:

- NDB Cluster: REPLACE statements did not work correctly on an NDB table having both a primary key and a unique key. In such cases, proper values were not set for columns which were not explicitly referenced in the statement. (Bug #20728)
- NDB Cluster: Trying to create or drop a table while a node was restarting caused the node to crash. This is now handled by raising an error. (Bug #18781)
- NDB Cluster: Running **ndbd** --nowait-nodes=id where *id* was the node ID of a node that was already running would fail with an invalid error message. (Bug #20419)
- NDB Cluster: Incorrect values were inserted into AUTO_INCREMENT columns of tables restored from a cluster backup. (Bug #20820)
- NDB Cluster: When attempting to restart the cluster following a data import, the cluster would fail during Phase 4 of the restart with Error 2334: Job buffer congestion. (Bug #20774)
- NDB Cluster: A node failure during a scan could sometime cause the node to crash when restarting too quickly following the failure. (Bug #20197)
- NDB Cluster: It was possible to use port numbers greater than 65535 for ServerPort in the config.ini file. (Bug #19164)
- NDB Cluster: Under certain circumstances, a node that was shut down then restarted could hang during the restart. (Bug #18863)
- NDB Cluster (Replication): In some cases, a large number of MySQL servers sending requests to the cluster simultaneously could cause the cluster to crash. This could also be triggered by many NDB API clients making simultaneous event subscriptions or unsubscriptions. (Bug #20683)

- NDB Cluster (NDB API): `NdbScanOperation::readTuples()` and `NdbIndexScanOperation::readTuples()` ignored the *batch* parameter. (Bug #20252)
- The use of `WHERE col_name IS NULL` in `SELECT` statements reset the value of `LAST_INSERT_ID()` to zero. (Bug #14553)
- The server crashed when using the range access method to execute a subquery with a `ORDER BY DESC` clause. (Bug #20869)
- Use of the join cache in favor of an index for `ORDER BY` operations could cause incorrect result sorting. (Bug #17212)
- A user-defined function that is called on each row of a returned result set, could receive an `is_null` state that is set, if it was set previously. Now, the `is_null` state is reset to false before each invocation of a UDF. (Bug #19904)
- Referring to a stored function qualified with the name of one database and tables in another database caused a “table doesn't exist” error. (Bug #18444)
- For NDB and possibly InnoDB tables, a `BEFORE UPDATE` trigger could insert incorrect values. (Bug #18437)
- Triggers on tables in the `mysql` database caused a server crash. Triggers for tables in this database now are disallowed. (Bug #18361)
- The length of the pattern string prefix for `LIKE` operations was calculated incorrectly for multi-byte character sets. As a result, the scanned range was wider than necessary if the prefix contained any multi-byte characters, and rows could be missing from the result set. (Bug #16674, Bug #18359)
- For very complex `SELECT` statements could create temporary tables that were too big, but for which the temporary files did not get removed, causing subsequent queries to fail. (Bug #11824)
- For spatial data types, the server formerly returned these as `VARSTRING` values with a binary collation. Now the server returns spatial values as `BLOB` values. (Bug #10166)

- Using `SELECT` and a table join while running a concurrent `INSERT` operation would join incorrect rows. (Bug #14400)
- Using `SELECT` on a corrupt table using the dynamic record format could cause a server crash. (Bug #19835)
- Using tables from MySQL 4.x in MySQL 5.x, in particular those with `VARCHAR` fields and using `INSERT DELAYED` to update data in the table would result in either data corruption or a server crash. (Bug #16611, Bug #16218, Bug #17294)
- Checking a spatial table (using `CHECK TABLE`) with an index and only one row would indicate a table corruption. (Bug #17877)
- `SHOW GRANTS FOR CURRENT_USER` did not return definer grants when executed in `DEFINER` context (such as within a stored procedure defined with `SQL SECURITY DEFINER`), it returned the invoker grants. (Bug #15298)
- For `SELECT ... FOR UPDATE` statements that used `DISTINCT` or `GROUP BY` over all key parts of a unique index (or primary key), the optimizer unnecessarily created a temporary table, thus losing the linkage to the underlying unique index values. This caused a `Result set not updatable` error. (The temporary table is unnecessary because under these circumstances the distinct or grouped columns must also be unique.) (Bug #16458)
- The first time a user who had been granted the `CREATE ROUTINE` privilege used that privilege to create a stored function or procedure, the `Password` column in that user's row in the `mysql.user` table was set to `NULL`. (Bug #19857)
- Creation of a view as a join of views or tables could fail if the views or tables are in different databases. (Bug #20482)
- Use of `MIN()` or `MAX()` with `GROUP BY` on a `ucs2` column could cause a server crash. (Bug #20076)
- `INSERT INTO ... SELECT ... LIMIT 1` could be slow because the `LIMIT` was ignored when selecting candidate rows. (Bug #9676)

- Certain queries having a `WHERE` clause that included conditions on multi-part keys with more than 2 key parts could produce incorrect results and send [Note] `Use_count: Wrong count for key at...` messages to `STDERR`. (Bug #16168)
- The `mysql_list_fields()` C API function returned the incorrect table name for views. (Bug #19671)
- A cast problem caused incorrect results for prepared statements that returned float values when MySQL was compiled with `gcc 4.0`. (Bug #19694)

D.1.2. Changes in release 5.0.24 (Not yet released)

This is a bugfix release for the current production release family.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Network* (a commercial MySQL offering). For more details please see <http://www.mysql.com/network/advisors.html>.

Bugs fixed:

- MySQL 5.0.23 contained a fix for Bug #10952 that has been reverted in 5.0.24 because it introduced the risk of unintended data loss.
- The `FEDERATED` storage engine did not allow creation of `UNIQUE` indexes on nullable columns. (Bug #15133)
- A `SELECT` that used a subquery in the `FROM` clause that did not select from a table failed when the subquery was used in a join. (Bug #21002)
- `REPLACE ... SELECT` for a view required the `INSERT` privilege for tables other than the table being modified. (Bug #21135)
- Failure to account for a `NULL` table pointer on big-endian machines could cause a server crash during type conversion. (Bug #21135)

- **mysqldump** sometimes did not select the correct database before trying to dump views from it, resulting in an empty result set that caused **mysqldump** to die with a segmentation fault. (Bug #21014)

D.1.3. Changes in release 5.0.23 (Not released)

MySQL 5.0.23 was never officially released.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Network* (a commercial MySQL offering). For more details please see <http://www.mysql.com/network/advisors.html>.

Functionality added or changed:

- **NDB Cluster**: The limit of 2048 ordered indexes per cluster has been lifted. There is now no upper limit on the number of ordered indexes (including `AUTO_INCREMENT` columns) that may be used. (Bug #14509)
- **NDB Cluster**: The status variables `Ndb_connected_host` and `Ndb_connected_port` were renamed to `Ndb_config_from_host` and `Ndb_config_from_port`, respectively.
- The **mysql_upgrade** command has been converted from a shell script to a C program, so it is available on non-Unix systems such as Windows. This program should be run for each MySQL upgrade. See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).
- Binary distributions that include SSL support now are built using yaSSL when possible.
- Added the `--ssl-verify-server-cert` option to MySQL client programs. This option causes the server's Common Name value in its certificate to be verified against the hostname used when connecting to the server, and the connection is rejected if there is a mismatch. Added `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` option for the `mysql_options()` C API function to enable this verification. This feature can be used to prevent

man-in-the-middle attacks. Verification is disabled by default. (Bug #17208)

- Added the `ssl_ca`, `ssl_capath`, `ssl_cert`, `ssl_cipher`, and `ssl_key` system variables, which display the values given via the corresponding command options. See [Section 5.9.7.3, “SSL Command Options”](#). (Bug#19606)
- Added the `log_queries_not_using_indexes` system variable. (Bug#19616)
- Added the `--angel-pid-file` option to **mysqlmanager** for specifying the file in which the angel process records its process ID when **mysqlmanager** runs in daemon mode. (Bug #14106)
- The `ONLY_FULL_GROUP_BY` SQL mode now also applies to the `HAVING` clause. That is, columns not named in the `GROUP BY` clause cannot be used in the `HAVING` clause if not used in an aggregate function. (Bug #18739)
- SQL syntax for prepared statements now supports `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE`. (Bug #19308)
- The bundled yaSSL library was upgraded to version 1.3.5. This improves handling of certain problems with SSL-related command options. (Bug #17737)
- Added the `--set-charset` option to **mysqlbinlog** to allow the character set to be specified for processing binary log files. (Bug #18351)
- For a table with an `AUTO_INCREMENT` column, `SHOW CREATE TABLE` now shows the next `AUTO_INCREMENT` value to be generated. (Bug #19025)
- It is now possible to use `NEW.var_name` values within triggers as `INOUT` parameters to stored procedures. (Bug #14635)
- The **mysqldumpslow** script has been moved from client RPM packages to server RPM packages. This corrects a problem where **mysqldumpslow** could not be used with a client-only RPM install, because it depends on **my_print_defaults** which is in the server RPM. (Bug #20216)

Bugs fixed:

- **mysqldump** would not dump views that had become invalid because a table named in the view definition had been dropped. Instead, it quit with an error message. Now you can specify the `--force` option to cause **mysqldump** to keep going and write a SQL comment containing the view definition to the dump output. (Bug #17371)
- The `WITH CHECK OPTION` was not enforced when a `REPLACE` statement was executed against a view. (Bug #19789)
- The use of `MIN()` and `MAX()` on columns with a partial index produced incorrect results in some queries. (Bug #18206)
- Concatenating the results of multiple constant subselects produced incorrect results. (Bug #16716)
- A “table not found” error could occur for statements that called a function defined in another database. (Bug #17199)
- A buffer overwrite error in Instance Manager caused a crash. (Bug #20622)
- Re-execution of a prepared multiple-table `DELETE` statement that involves a trigger or stored function can result in a server crash. (Bug #19634)
- On Windows, corrected a crash stemming from differences in Visual C runtime library routines from POSIX behavior regarding invalid file descriptors. (Bug #18275)
- Multiple-table updates with `FEDERATED` tables could cause a server crash. (Bug #19773)
- On Windows, terminating **mysqld** with Control-C could result in a crash during shutdown. (Bug #18235)
- On Windows, removal of binary log files would fail if the files were already open. (Bug #19208)
- **mysqldump** produced garbled output for view definitions. (Bug #18462)

- The omission of leading zeros in dates could lead to erroneous results when these were compared with the output of certain date and time functions. (Bug #16377)
- An invalid comparison between keys in partial indexes over multi-byte character fields could lead to incorrect result sets if the selected query execution plan used a range scan by a partial index over a UTF8 character field. This also caused incorrect results under similar circumstances with many other character sets. (Bug #14896)
- NDB cluster: Cluster system status variables were not updated. (Bug #11459)
- NDB cluster: The cluster's data nodes would fail while trying to load data when `NoOfFrangmentLogFiles` was equal to 1. (Bug #19894)
- NDB cluster: A problem with error handling when `ndb_use_exact_count` was enabled could lead to incorrect values returned from queries using `COUNT()`. A warning is now returned in such cases. (Bug #19202)
- NDB cluster: Restoring a backup made using **ndb_restore** failed when the backup had been taken from a cluster whose data memory was full. (Bug #19852)
- NDB cluster: TEXT columns in Cluster tables having both an explicit primary key and a unique key were not correctly updated by REPLACE statements. (Bug #19906)
- NDB cluster: An internal formatting error caused some management client error messages to be unreadable. (Bug #20016)
- NDB cluster: Running management client commands while **mgmd** was in the process of disconnecting could cause the management server to fail. (Bug #19932)
- NDB cluster (NDBAPI): Update operations on blobs were not checked for illegal operations.

Note: Read locks with blob update operations are now upgraded from read committed to read shared.

- NDB Cluster: The management client ALL STOP command shut down **mgmd** processes (as well as **ndbd** processes). (Bug #18966)
- NDB Cluster: LOAD DATA LOCAL failed to ignore duplicate keys in Cluster tables. (Bug #19496)
- NDB Cluster: Repeated CREATE - INSERT - DROP operations tables could in some circumstances cause the MySQL table definition cache to become corrupt, so that some **mysqld** processes could access table information but others could not. (Bug #18595)
- NDB Cluster: The **mgm** client command ALL CLUSTERLOG STATISTICS=15; had no effect. (Bug #20336)
- NDB Cluster: TRUNCATE TABLE failed to reset the AUTO_INCREMENT counter. (Bug #18864)
- NDB Cluster: SELECT ... FOR UPDATE failed to lock the selected rows. (Bug #18184)
- NDB Cluster: The failure of a data node when preparing to commit a transaction (that is, while the node's status was CS_PREPARE_TO_COMMIT) could cause the failure of other cluster data nodes. (Bug #20185)
- NDB Cluster: Renaming a table in such a way as to move it to to a different database failed to move the table's indexes. (Bug #19967)
- NDB Cluster: Resources for unique indexes on Cluster table columns were incorrectly allocated, so that only one-fourth as many unique indexes as indicated by the value of UniqueHashIndexes could be created. (Bug #19623)
- NDB Cluster: Running ALL START in the NDB management client or restarting multiple nodes simultaneously could under some circumstances cause the cluster to crash. (Bug #19930)
- NDB Cluster: SELECT statements with a BLOB or TEXT column in the selected column list and a WHERE condition including a primary key lookup on a VARCHAR primary key produced empty result sets. **Note:** This issue affected the 5.0 series of MySQL Cluster releases only. (Bug #19956)

- NDB Cluster (NDBAPI): On big-endian platforms, `NdbOperation::write_attr()` did not update 32-bit fields correctly. (Bug #19537)
- NDB Cluster: Some queries having a `WHERE` clause of the form `c1=val1 OR c2 LIKE 'val2'` were not evaluated correctly. (Bug # 17421)
- NDB Cluster: Using “stale” **mysqld** .FRM files could cause a newly-restored cluster to fail. This situation could arise when restarting a MySQL Cluster using the `--initial` option while leaving connected **mysqld** processes running. (Bug #16875)
- NDB Cluster: Repeated use of the `SHOW` and `ALL STATUS` commands in the **ndb_mgm** client could cause the **mgmd** process to crash. (Bug #18591)
- NDB Cluster: An issue with **ndb_mgmd** prevented more than 27 **mysqld** processes from connecting to a single cluster at one time. (Bug #17150)
- NDB Cluster: Data node failures could cause excessive CPU usage by **ndb_mgmd**. (Bug #13987)
- NDB Cluster: `TRUNCATE` failed on tables having `BLOB` or `TEXT` columns with the error `Lock wait timeout exceeded`. (Bug #19201)
- NDB Cluster: Stopping multiple nodes could cause node failure handling not to be completed. (Bug #19039)
- NDB Cluster: **ndbd** could sometimes fail to start with the error `Node failure handling not completed following a graceful restart`. (Bug #18550)
- NDB Cluster: Backups could fail for large clusters with many tables, where the number of tables approached `MaxNumberOfTables`. (Bug #17607)
- On Windows, temporary tables containing `‘:’` in the name could not be created. (Bug #20616)
- The `--core-file-size` option for **mysqld_safe** was effective only for `root`. (Bug #17353)
- Some queries that used `ORDER BY` and `LIMIT` performed quickly in MySQL

3.23, but slowly in MySQL 4.x/5.x due to an optimizer problem. (Bug #4981)

- **mysql_upgrade** was missing from binary MySQL distributions. (Bug #18516, Bug #20403)
- Queries using an indexed column as the argument for the `MIN()` and `MAX()` functions following an `ALTER TABLE ... DISABLE KEYS` statement returned Got error 124 from storage engine until `ALTER TABLE ... ENABLE KEYS` was run on the table. (Bug #20357)
- A number of dependency issues in the RPM bench and test packages caused installation of these packages to fail. (Bug #20078)
- Nested natural joins worked executed correctly when executed as a non-prepared statement could fail with an Unknown column 'col_name' in 'field list' error when executed as a prepared statement, due to a name resolution problem. (Bug #15355)
- `GROUP BY` on an expression that contained a cast to `DECIMAL` produced an incorrect result. (Bug #19667)
- The `max_length` metadata value for columns created from `CONCAT()` could be incorrect when the collation of an argument differed from the collation of the `CONCAT()` itself. In some contexts such as `UNION`, this could lead to truncation of the column contents. (Bug #15962)
- The `MD5()` and `SHA()` functions treat their arguments as case-sensitive strings. But when they are compared, their arguments were compared as case-insensitive strings, which leads to two function calls with different arguments (and thus different results) compared as being identical. This can lead to a wrong decision made in the range optimizer and thus to an incorrect result set. (Bug #15351)
- For `BOOLEAN` mode full-text searches on non-indexed columns, `NULL` rows generated by a `LEFT JOIN` caused incorrect query results. (Bug #14708)
- `BIT` columns in a table could cause joins that use the table to fail. (Bug #18895)

- A UNION over more than 128 SELECT statements that use an aggregate function failed. (Bug #18175)
- InnoDB unlocked its data directory before committing a transaction, potentially resulting in non-recoverable tables if a server crash occurred before the commit. (Bug #19727)
- Multiple-table DELETE statements containing a subquery that selected from one of the tables being modified caused a server crash. (Bug #19225)
- With settings of read_buffer_size >= 2G and read_rnd_buffer_size >=2G, LOAD DATA INFILE failed with no error message or caused a server crash for files larger than 2GB. (Bug #12982)
- REPLACE statements caused activation of UPDATE triggers, not DELETE and INSERT triggers. (Bug #13479)
- The thread for INSERT DELAYED rows was maintaining a separate AUTO_INCREMENT counter, resulting in incorrect values being assigned if DELAYED and non-DELAYED inserts were mixed. (Bug #20195)
- **mysqldump** wrote an extra pair of DROP DATABASE and CREATE DATABASE statements if run with the --add-drop-database option and the database contained views. (Bug #17201)
- On 64-bit Windows systems, REGEXP for regular expressions with exactly 31 characters did not work. (Bug #19407)
- For **mysqld**, Valgrind revealed problems that were corrected: A dangling stack pointer being overwritten (Bug #20769); possible uninitialized data in a string comparison (Bug #20783); memory corruption in replication slaves when switching databases (Bug #19022); syscall write parameter pointing to uninitialized byte (Bug #20579).
- For **ndb_mgmd**, Valgrind revealed problems that were corrected: A memory leak (Bug #19318); a dependency on an uninitialized variable (Bug #20333).
- An update that used a join of a table to itself and modified the table on both sides of the join reported the table as crashed. (Bug #18036)

- SSL connections using yaSSL on OpenBSD could fail. (Bug #19191)
- On Windows, multiple clients simultaneously attempting to perform ALTER TABLE operations on an InnoDB table could deadlock. (Bug #17264)
- The `fill_help_tables.sql` file did not load properly if the ANSI_QUOTES SQL mode was enabled. (Bug #20542)
- The `fill_help_tables.sql` file did not contain a SET NAMES 'utf8' statement to indicate its encoding. This caused problems for some settings of the MySQL character set such as big5. (Bug #20551)
- The MySQL server startup script `/etc/init.d/mysql` (created from **mysql.server**) is now marked to ensure that the system services **ybind**, **nscd**, **ldap**, and **NTP** are started first (if these are configured on the machine). (Bug #18810)
- MERGE tables did not work reliably with BIT columns. (Bug #19648)
- For a reference to a non-existent index in FORCE INDEX, the error message referred to a column, not an index. (Bug #17873)
- Some yaSSL public function names conflicted with those from OpenSSL, causing conflicts for applications that linked against both OpenSSL and a version of `libmysqlclient` that was built with yaSSL support. The yaSSL public functions now are renamed to avoid this conflict. (Bug #19575)
- CHECK TABLE temporarily cleared the AUTO_INCREMENT value. Because it runs with a read lock, other readers could perform concurrent inserts, and if so, they could get an incorrect AUTO_INCREMENT value. CHECK TABLE no longer modifies the AUTO_INCREMENT value. (Bug #19604)
- If there is a global read lock, CREATE DATABASE, RENAME DATABASE, and DROP DATABASE could deadlock. (Bug #19815)
- On Linux, `libmysqlclient` when compiled with yaSSL using the **icc** compiler had a spurious dependency on C++ libraries. (Bug #20119)
- Using `CONCAT(@user_var, col_name)`, where `col_name` is a column in an INFORMATION_SCHEMA table, could cause erroneous duplication of data in the

query result. (Bug #19599)

- Results from `INFORMATION_SCHEMA.SCHEMATA` could contain uppercase information when `lower_case_table_names` was not 0. (Bug #17661)
- Grant table modifications sometimes did not refresh the in-memory tables if the hostname was `' '` or not specified. (Bug #16297)
- Invalid escape sequences in option files caused MySQL programs that read them to abort. (Bug #15328)
- InnoDB did not increment the `handler_read_prev` counter. (Bug #19542)
- Race conditions on certain platforms could cause the Instance Manager to fail to initialize. (Bug #19391)
- `ALTER TABLE` on a table created prior to 5.0.3 would cause table corruption if the `ALTER TABLE` did one of the following:
 - Change the default value of a column.
 - Change the table comment.
 - Change the table password.

(Bug #17001)

- An `ALTER TABLE` operation that does not need to copy data, when executed on a table created prior to MySQL 4.0.25, could result in a server crash for subsequent accesses to the table. (Bug #19192)
- The binary log lacked character set information for table name when dropping temporary tables. (Bug #14157)
- A B-TREE index on a MEMORY table erroneously reported duplicate entry error for multiple NULL values. (Bug #12873)
- Race conditions on certain platforms could cause the Instance Manager to try to restart the same instance multiple times. (Bug #18023)
- A `CREATE TABLE` statement that created a table from a materialized view did

not inherit default values from the underlying table. (Bug #19089)

- The `COM_STATISTICS` command was changed in 5.0.3 to display session status variable values rather than global values. This causes **mysqladmin status** information not to be useful for the `SLOW` queries and `OPENS` values. Now `COM_STATISTICS` displays the global values for `SLOW` queries and `OPENS`. (Bug #18669)
- `INFORMATION_SCHEMA.TABLES` provided inconsistent info about invalid views. This could cause server crashes or result in incorrect data being returned for queries that attempt to obtain information from `INFORMATION_SCHEMA` tables about views using stored functions. (Bug #18282)
- Multiple calls to a stored procedure that selects from `INFORMATION_SCHEMA` could cause a server crash. (Bug #17204)
- Premature optimization of nested subqueries in the `FROM` clause that refer to aggregate functions could lead to incorrect results. (Bug #19077)
- A view definition that referred to an alias in the `HAVING` clause could be saved in the `.frm` file with the alias replaced by the expression that it referred to, causing failure of subsequent `SELECT * FROM view_name` statements. (Bug #19573)
- Several aspects of view privileges were being checked incorrectly. (Bug #18681, Bug #20363)
- A view with a non-existent account in the `DEFINER` clause caused `SHOW CREATE VIEW` to fail. Now `SHOW CREATE VIEW` issues a warning instead. (Bug #20048)
- A bug in NTPL threads on Linux could result in a deadlock with `FLUSH TABLES WITH READ LOCK` under some conditions. (Bug #20048)
- MyISAM table deadlock was possible if one thread issued a `LOCK TABLES` request for write locks and then an administrative statement such as `OPTIMIZE TABLE`, if between the two statements another client meanwhile issued a multiple-table `SELECT` for some of the locked tables. (Bug #16986)

- Subqueries that produced a `BIGINT UNSIGNED` value were being treated as returning a signed value. (Bug #19700)
- The patch for Bug #17164 introduced the problem that some outer joins were incorrectly converted to inner joins. (Bug #19816)
- `BLOB` or `TEXT` arguments to or values returned from stored functions were not copied properly if too long and could become garbled. (Bug #18587)
- Selecting data from a `MEMORY` table with a `VARCHAR` column and a `HASH` index over it returned only the first row matched. (Bug #18233)
- `CREATE TABLE ... SELECT` did not always produce the proper column default value in `TRADITIONAL SQL` mode. (Bug #17626)
- Privilege checking on the contents of the `INFORMATION_SCHEMA.VIEWS` table was insufficiently restrictive. (Bug #16681)
- The result from `CONV()` is a string, but was not always treated the same way as a string when converted to a real value for an arithmetic operation. (Bug #13975)
- `CREATE TABLE ... SELECT ...` statements that used a stored function explicitly or implicitly (through a view) resulted in a `Table not locked` error. (Bug #12472, Bug #15137)
- Within a trigger, `SET` used the `SQL` mode of the invoking statement, not the mode in effect at trigger creation time. (Bug #6951)
- The server no longer uses a signal handler for signal 0 because it could cause a crash on some platforms. (Bug #15869)
- Revised memory allocation for local objects within stored functions and triggers to avoid memory leak for repeated function or trigger invocation. (Bug #17260)
- `EXPLAIN ... SELECT INTO` caused the client to hang. (Bug #15463)
- Symlinking `.mysql_history` to `/dev/null` to suppress statement history saving by **mysql** did not work. (**mysql** deleted the symlink and recreated

.mysql_history as a regular file, and then wrote history to it.) (Bug #16803)

- The basedir and tmpdir system variables could not be accessed via @@var_name syntax. (Bug #1039)
- For certain CREATE VIEW statements, the server did not detect invalid subqueries within the SELECT part. (Bug #7549)
- The range operator failed and caused a server crash for clauses of the form tbl_name.unsigned_keypart NOT IN (negative_const, ...). (Bug #19618)
- Returning the value of a system variable from a stored function caused a server crash. (Bug #18037)
- Updates to a MEMORY table caused the size of BTREE indexes for the table to increase. (Bug #18160)
- REPAIR TABLE did not restore the length for packed keys in tables created under MySQL 4.x. (Bug #17810)
- Selecting from a view that used GROUP BY on a non-constant temporal interval (such as DATE(col) + INTERVAL TIME_TO_SEC(col) SECOND could cause a server crash. (Bug #19490)
- An outer join of two views that was written using { OJ ... } syntax could cause a server crash. (Bug #19396)
- LOAD_FILE() returned an error if the file did not exist, rather than NULL as it should according to the manual. (Bug #10418)
- For certain CREATE TABLE ... SELECT statements, the selected values were truncated when inserted into the new table. (Bug #17048)
- Use of uninitialized user variables in a subquery in the FROM clause results in bad entries in the binary log. (Bug #19136)
- In the INFORMATION_SCHEMA.COLUMNS table, the values for the CHARACTER_MAXIMUM_LENGTH and CHARACTER_OCTET_LENGTH columns were incorrect for multi-byte character sets. (Bug #19236)

- An entry in the `mysql.proc` table with an empty routine name caused access to the `INFORMATION_SCHEMA.ROUTINES` table to crash the server. (Bug #18177)
- A range access optimizer heuristic was invalid, causing some queries to be much slower in MySQL 5.0 than in 4.0. (Bug #17379, Bug #18940)
- `IS_USED_LOCK()` could return an incorrect connection identifier. (Bug #16501)
- **mysql** displayed `NULL` for strings that are empty or contain only spaces. (Bug #19564)
- Concurrent reading and writing of privilege structures could crash the server. (Bug #16372)
- A `NUL` byte within a comment in a statement string caused the rest of the string not to be written to the query log, allowing logging to be bypassed. ([CVE-2006-0903](#)) (Bug #17667)
- **mysql-test-run.pl** started `NDB` even for test cases that didn't need it. (Bug #19083)
- `SELECT DISTINCT` queries sometimes returned only the last row. (Bug #18068)
- Use of `CONVERT_TZ()` in a stored function or trigger (or in a stored procedure called from a stored function or trigger) caused an error. (Bug #11081)
- Some queries were slower in 5.0 than in 4.1 because some 4.1 cost-evaluation code had not been merged into 5.0. (Bug #14292)
- Index prefixes for `utf8 VARCHAR` columns did not work for `UPDATE` statements. (Bug #19080)
- `InnoDB` does not support `SPATIAL` indexes, but did not prevent creation of such an index. (Bug #15860)
- The configuration information for building the embedded server on

Windows was missing a file. (Bug #18455)

- The parser leaked memory when its stack needed to be extended. (Bug #18930)
- When **myisamchk** needed to rebuild a table, AUTO_INCREMENT information was lost. (Bug #10405)
- LOAD DATA FROM MASTER would fail when trying to load the INFORMATION_SCHEMA database from the master, because the INFORMATION_SCHEMA system database would already exist on the slave. (Bug #18607)
- The binary log would create an incorrect DROP query when creating temporary tables during replication. (Bug #17263)
- The IN-to-EXISTS transformation was making a reference to a parse tree fragment that was left out of the parse tree. This caused problems with prepared statements. (Bug #18492)
- In **mysqltest**, --sleep=0 had no effect. Now it correctly causes sleep commands in test case files to sleep for 0 seconds. (Bug #18312)
- Attempting to set the default value of an ENUM or SET column to NULL caused a server crash. (Bug #19145)
- The sql_notes and sql_warnings system variables were not always displayed correctly by SHOW VARIABLES (for example, they were displayed as ON after being set to OFF). (Bug #16195)
- The sql_big_selects system variable was not displayed by SHOW VARIABLES. (Bug #17849)
- The system_time_zone and version_* system variables could not be accessed via SELECT @@var_name syntax. (Bug #12792, Bug #15684)
- Flushing the compression buffer (via FLUSH TABLE) no longer increases the size of an unmodified ARCHIVE table. (Bug #19204)
- RPM packages had spurious dependencies on Perl modules and other

programs. (Bug #13634)

D.1.4. Changes in release 5.0.22 (24 May 2006)

This is a security fix release for the previous production release family.

This release includes the security fix described later in this section and a few other changes to resolve build problems, relative to the last official MySQL release (5.0.21). If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Network* (a commercial MySQL offering). For more details please see <http://www.mysql.com/network/advisors.html>.

Bugs fixed:

- **Security fix:** An SQL-injection security hole has been found in multi-byte encoding processing. The bug was in the server, incorrectly parsing the string escaped with the `mysql_real_escape_string()` C API function. ([CVE-2006-2753](#), Bug#8378)

This vulnerability was discovered and reported by Josh Berkus <josh@postgresql.org> and Tom Lane <tgl@sss.pgh.pa.us> as part of the inter-project security collaboration of the OSDB consortium. For more information about SQL injection, please see the following text.

Discussion: An SQL-injection security hole has been found in multi-byte encoding processing. An SQL-injection security hole can include a situation whereby when a user supplied data to be inserted into a database, the user might inject SQL statements into the data that the server will execute. With regards to this vulnerability, when character set unaware-escaping is used (for example, `addslashes()` in PHP), it is possible to bypass the escaping in some multi-byte character sets (for example, SJIS, BIG5 and GBK). As a result, a function such as `addslashes()` is not able to prevent SQL-injection attacks. It is impossible to fix this on the server side. The best solution is for applications to use character set-aware escaping offered by a function such `mysql_real_escape_string()`.

However, a bug was detected in how the MySQL server parses the output of `mysql_real_escape_string()`. As a result, even when the character set-

aware function `mysql_real_escape_string()` was used, SQL injection was possible. This bug has been fixed.

Workarounds: If you are unable to upgrade MySQL to a version that includes the fix for the bug in `mysql_real_escape_string()` parsing, but run MySQL 5.0.1 or higher, you can use the `NO_BACKSLASH_ESCAPES` SQL mode as a workaround. (This mode was introduced in MySQL 5.0.1.) `NO_BACKSLASH_ESCAPES` enables an SQL standard compatibility mode, where backslash is not considered a special character. The result will be that queries will fail.

To set this mode for the current connection, enter the following SQL statement:

```
SET sql_mode='NO_BACKSLASH_ESCAPES';
```

You can also set the mode globally for all clients:

```
SET GLOBAL sql_mode='NO_BACKSLASH_ESCAPES';
```

This SQL mode also can be enabled automatically when the server starts by using the command-line option `--sql-mode=NO_BACKSLASH_ESCAPES` or by setting `sql-mode=NO_BACKSLASH_ESCAPES` in the server option file (for example, `my.cnf` or `my.ini`, depending on your system).

- The patch for Bug #8303 broke the fix for Bug #8378 and was undone. (In string literals with an escape character (`\`) followed by a multi-byte character that has a second byte of (`\`), the literal was not interpreted correctly. The next byte now is escaped, not the entire multi-byte character. This means it a strict reverse of the `mysql_real_escape_string()` function.)
- The client libraries had not been compiled for position-independent code on Solaris-SPARC and AMD x86_64 platforms. (Bug #13159, Bug #14202, Bug #18091)
- Running **mysampack** followed by **mysamchk** with the `--unpack` option would corrupt the `auto_increment` key. (Bug #12633)

D.1.5. Changes in release 5.0.21 (02 May 2006)

This is a bugfix release for the current production release family.

This MySQL 5.0.21 release includes the patches for recently reported security vulnerabilities in the MySQL client-server protocol. We would like to thank Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting these to us.

This section documents all changes and bug fixes that have been applied since the last official MySQL release. If you would like to receive more fine-grained and personalized *update alerts* about fixes that are relevant to the version and features you use, please consider subscribing to *MySQL Network* (a commercial MySQL offering). For more details please see <http://www.mysql.com/network/advisors.html>.

Functionality added or changed:

- **Security enhancement:** Added the global `max_prepared_stmt_count` system variable to limit the total number of prepared statements in the server. This limits the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. The current number of prepared statements is available through the `prepared_stmt_count` system variable. (Bug #16365)
- The `MySQL-shared-compat-5.0.X-i386.rpm` shared compatibility RPMs no longer contain libraries for MySQL 5.1. This avoids a conflict because the 5.0 and 5.1 libraries share the same soname number. It contains libraries for 3.23, 4.0, 4.1, and 5.0. (Bug #19288)
- Creating a table in an InnoDB database with a column name that matched the name of an internal InnoDB column (including `DB_ROW_ID`, `DB_TRX_ID`, `DB_ROLL_PTR` and `DB_MIX_ID`) would cause a crash. MySQL now returns error 1005 (cannot create table) with `errno` set to -1. (Bug #18934)
- **NDB Cluster:** It is now possible to perform a partial start of a cluster. That is, it is now possible to bring up the cluster without running `ndbd --initial` on all configured data nodes first. (Bug #18606)
- **NDB Cluster:** A new `--nowait-nodes` startup option for **ndbd** makes it possible to “skip” specific nodes without waiting for them to start when starting the cluster. See [Section 15.6.5.2, “Command Options for ndbd”](#).

- **NDB Cluster:** It is now possible to install MySQL with Cluster support to a non-default location and change the search path for font description files using either the `--basedir` or `--character-sets-dir` options. (Previously in MySQL 5.0, **ndbd** searched only the default path for character sets.)
- In result set metadata, the `MYSQL_FIELD.length` value for BIT columns now is reported in number of bits. For example, the value for a `BIT(9)` column is 9. (Formerly, the value was related to number of bytes.) (Bug #13601)
- The default for the `innodb_thread_concurrency` system variable was changed to 8. (Bug #15868)

Bugs fixed:

- **Security fix:** A malicious client, using specially crafted invalid login or `COM_TABLE_DUMP` packets was able to read uninitialized memory, which potentially, though unlikely in MySQL, could have led to an information disclosure. ([CVE-2006-1516](#), [CVE-2006-1517](#)) Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting this bug.
- **Security fix:** A malicious client, using specially crafted invalid `COM_TABLE_DUMP` packets was able to trigger an exploitable buffer overflow on the server. ([CVE-2006-1518](#)) Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and reporting this bug.
- **Security fix:** Invalid arguments to `DATE_FORMAT()` caused a server crash. ([CVE-2006-3469](#), Bug #20729) Thanks to Jean-David Maillefer for discovering and reporting this problem to the Debian project and to Christian Hammers from the Debian Team for notifying us of it.
- **NDB Cluster:** A simultaneous `DROP TABLE` and table update operation utilising a table scan could trigger a node failure. (Bug #18597)
- **mysql-test-run** could not be run as root. (Bug #17002)
- `MySQL-shared-compat-5.0.13-0.i386.rpm`, `MySQL-shared-compat-5.0.15-0.i386.rpm`, `MySQL-shared-compat-5.0.18-0.i386.rpm`, `MySQL-shared-compat-5.0.19-0.i386.rpm`, `MySQL-shared-compat-5.0.20-0.i386.rpm`, and `MySQL-shared-compat-5.0.20a-0.i386.rpm` incorrectly depended on `glibc 2.3` and could not be installed on a `glibc 2.2` system.

(Bug #16539)

- IA-64 RPM packages for Red Hat and SuSE Linux that were built with the `icc` compiler incorrectly depended on `icc` runtime libraries. (Bug #16662)
- After calling `FLUSH STATUS`, the `max_used_connections` variable did not increment for existing connections and connections which use the thread cache. (Bug #15933)
- MySQL would not compile on Linux distributions that use the `tinfo` library. (Bug #18912)
- Within a trigger, `CONNECTION_ID()` did not return the connection ID of the thread that caused the trigger to be activated. (Bug #16461)
- The `yaSSL` library returned a cipher list in a manner incompatible with `OpenSSL`. (Bug #18399)
- For single-`SELECT` union constructs of the form `(SELECT ... ORDER BY order_list1 [LIMIT n]) ORDER BY order_list2`, the `ORDER BY` lists were concatenated and the `LIMIT` clause was ignored. (Bug #18767)
- `CREATE VIEW` statements would not be replicated to the slave if the `--replicate-wild-ignore-table` rule was enabled. (Bug #18715)
- Index corruption could occur in cases when `key_cache_block_size` was not a multiple of `myisam_block_size` (for example, with `key_cache_block_size=1536` and `myisam_block_size=1024`). (Bug #19079)
- `LAST_INSERT_ID()` in a stored function or trigger returned zero. . (Bug #15728)
- Use of `CONVERT_TZ()` in a view definition could result in spurious syntax or access errors. (Bug #15153)
- `UNCOMPRESS(NULL)` could cause subsequent `UNCOMPRESS()` calls to return `NULL` for legal non-`NULL` arguments. (Bug #18643)
- Conversion of a number to a `CHAR UNICODE` string returned an invalid result.

(Bug #18691)

- DELETE and UPDATE statements that used large NOT IN (value_list) clauses could use large amounts of memory. (Bug #15872)
- Prevent recursive views caused by using RENAME TABLE on a view after creating it. (Bug #14308)
- A LOCK TABLES statement that failed could cause MyISAM not to update table statistics properly, causing a subsequent CHECK TABLE to report table corruption. (Bug #18544)
- For a reference to a non-existent stored function in a stored routine that had a CONTINUE handler, the server continued as though a useful result had been returned, possibly resulting in a server crash. (Bug #18787)
- InnoDB did not use a consistent read for CREATE ... SELECT when innodb_locks_unsafe_for_binlog was set. (Bug #18350)
- InnoDB could read a delete mark from its system tables incorrectly. (Bug #19217)
- Corrected a syntax error in **mysql-test-run.sh**. (Bug #19190)
- A missing DEBUG_RETURN() caused the server to emit a spurious error message: missing DEBUG_RETURN or DEBUG_VOID_RETURN macro in function "open_table". (Bug #18964)
- DROP DATABASE did not drop stored routines associated with the database if the database name was longer than 21 characters. (Bug #18344)
- Avoid trying to include <asm/atomic.h> when it doesn't work in C++ code. (Bug #13621)
- Executing SELECT on a large table that had been compressed within **myisampack** could cause a crash. (Bug #17917)
- NDB cluster: When attempting to create an index on a BIT or BLOB column, Error 743: Unsupported character set in table or index was returned instead of Error 906: Unsupported attribute type in index.

- Within stored routines, usernames were parsed incorrectly if they were enclosed within quotes. (Bug #13310)
- Casting a string to `DECIMAL` worked, but casting a trimmed string (using `LTRIM()` or `RTRIM()`) resulted in loss of decimal digits. (Bug #17043)
- `NDB cluster`: On slow networks or CPUs, the management client `SHOW` command could sometimes erroneously show all data nodes as being master nodes belonging to nodegroup 0. (Bug #15530)
- If the second or third argument to `BETWEEN` was a constant expression such as `'2005-09-01 - INTERVAL 6 MONTH` and the other two arguments were columns, `BETWEEN` was evaluated incorrectly. (Bug #18618)
- If the first argument to `BETWEEN` was a `DATE` or `TIME` column of a view and the other arguments were constants, `BETWEEN` did not perform conversion of the constants to the appropriate temporary type, resulting in incorrect evaluation. (Bug #16069)
- Server and clients ignored the `--sysconfdir` option that was passed to **configure**. (Bug #15069)
- `NDB cluster`: In a 2-node cluster with a node failure, restarting the node with a low value for `StartPartialTimeout` could cause the cluster to come up partitioned (“split-brain” issue). (Bug #16447)

A similar issue could occur when the cluster was first started with a sufficiently low value for this parameter. (Bug #18612)

- `NDB cluster`: On systems with multiple network interfaces, data nodes would get “stuck” in startup phase 2 if the interface connecting them to the management server was working on node startup while the interface interconnecting the data nodes experienced a temporary outage. (Bug #15695)
- `NDB cluster`: Unused open handlers for tables in which the metadata had changed were not properly closed. This could result in stale results from Cluster tables following an `ALTER TABLE`. (Bug #13228)
- `NDB cluster`: Uninitialized internal variables could lead to unexpected

results. (Bug #11033, Bug #11034)

- For InnoDB tables, an expression of the form `col_name BETWEEN col_name2 - INTERVAL x DAY AND col_name2 + INTERVAL x DAY` when used in a join returned incorrect results. (Bug #14360)
- `INSERT DELAYED` into a view caused an infinite loop. (Bug #13683)
- Lettercase in database name qualifiers was not consistently handled properly in queries when `lower_case_table_names` was set to 1. (Bug #15917)
- The optimizer could cause a server crash or use a non-optimal subset of indexes when evaluating whether to use Index Merge/Intersection variant of `index_merge` optimization. (Bug #19021)
- The presence of multiple equalities in a condition after reading a constant table could cause the optimizer not to use an index. This resulted in certain queries being much slower than in MySQL 4.1. (Bug #16504)
- A recent change caused the **mysql** client not to display `NULL` values correctly and to display numeric columns left-justified rather than right-justified. The problems have been corrected. (Bug #18265)
- `mysql_reconnect()` sent a `SET NAMES` statement to the server, even for pre-4.1 servers that do not understand the statement. (Bug #18830)
- `COUNT(*)` on a MyISAM table could return different results for the base table and a view on the base table. (Bug #18237)
- `DELETE` with `LEFT JOIN` for InnoDB tables could crash the server if `innodb_locks_unsafe_for_binlog` was enabled. (Bug #15650)
- InnoDB failure to release an adaptive hash index latch could cause a server crash if the query cache was enabled. (Bug #15758)
- For **mysql.server**, if the `basedir` option was specified after `datadir` in an option file, the setting for `datadir` was ignored and assumed to be located under `basedir`. (Bug #16240)

- The euro sign (€) was not stored correctly in columns using the latin1_german1_ci or latin1_general_ci collation. (Bug #18321)
- EXTRACT(QUARTER FROM date) returned unexpected results. (Bug #18100)
- TRUNCATE did not reset the AUTO_INCREMENT counter for MyISAM tables when issued inside a stored procedure. (Bug #14945)

Note: This bug did not affect InnoDB tables. Also, TRUNCATE does not reset the AUTO_INCREMENT counter for NDBCluster tables regardless of when it is called (see Bug #18864).

- The server was always built as though --with-extra-charsets=complex had been specified. (Bug #12076)
- A query using WHERE (column_1, column_2) IN ((value_1, value_2)[, (...), ...], ...) would return incorrect results. (Bug #16248)
- Queries of the form SELECT DISTINCT timestamp_column WHERE date_function(timestamp_col) = constant did not return all matching rows. (Bug #16710)
- When running a query that contained a GROUP_CONCAT(SELECT GROUP_CONCAT(...)), the result was NULL except in the ROLLUP part of the result, if there was one. (Bug #15560)
- For tables created in a MySQL 4.1 installation upgraded to MySQL 5.0 and up, multiple-table updates could update only the first matching row. (Bug #16281)
- NDB Cluster: When multiple node restarts were attempted without allowing each restart to complete, the error message returned was Array index out of bounds rather than Too many crashed replicas. (Bug #18349)
- CAST(double AS SIGNED INT) for large double values outside the signed integer range truncates the result to be within range, but the result sometimes had the wrong sign, and no warning was generated. (Bug #15098)
- Updating a field value when also requesting a lock with GET_LOCK() would

cause slave servers in a replication environment to terminate. (Bug #17284)

D.1.6. Changes in release 5.0.20a (18 April 2006)

This is a bugfix release for the current production release family. It replaces MySQL 5.0.20.

Changes from 5.0.20 to 5.0.20a:

- The fix for “Command line options are ignored for mysql client” (Bug #16855) has been revoked because it introduced an incompatible change in the way the **mysql** command-line client selects the server to connect to. In the worst case, this might have led to a client issuing commands to a server for which they were not intended, and this must not happen. To help all users in understanding this subject, [Section 4.2, “Invoking MySQL Programs”](#) now includes additional explanation of how command options with regard to host selection.
- The code of the `yassl` library has been improved to avoid the dependency on a C++ runtime library, so a link with pure C applications is now possible on additional (but not yet all) platforms. We are working on fixing the remaining issues.

Additional information about SSL support:

- With version 5.0.20a, SSL support is contained in all binaries for all Unix (including Linux) and Windows platforms except AIX, HP-UX, OpenServer 6, and the RPMs specific for RHAS3/RHAS4/SLES9 on Itanium CPUs (`ia64`); It is also not contained in those for Novell Netware. We are trying to add these platforms in future versions.
- Please note that the original 5.0.20 announcement included inexact wording: SSL support is “included” in both server and client, but by default not “enabled”. SSL can be enabled by passing the SSL-related options (`--ssl`, `--ssl-key=...`, `--ssl-cert=...`, `--ssl-ca=...`) when starting the server and the client or by specifying these options in an option file. For more information, see [Section 5.9.7, “Using Secure Connections”](#).

D.1.7. Changes in release 5.0.20 (31 March 2006)

Functionality added or changed:

- Added the `--sysdate-is-now` option to **mysqld** to enable `SYSDATE()` to be treated as an alias for `NOW()`. See [Section 12.5, “Date and Time Functions”](#). (Bug #15101)
- InnoDB: The InnoDB storage engine now provides a descriptive error message if `ibdata` file information is omitted from `my.cnf`. (Bug #16827)
- The NDBCluster storage engine now supports `INSERT IGNORE` and `REPLACE` statements. Previously, these statements failed with an error. (Bug #17431)
- Builds for Windows, Linux, and Unix (except AIX) platforms now have SSL support enabled, in the server as well as in the client libraries. Because part of the SSL code is written in C++, this does introduce dependencies on the system's C++ runtime libraries in several cases, depending on compiler specifics. (Bug #18195)
- The syntax for `CREATE PROCEDURE` and `CREATE FUNCTION` statements now includes a `DEFINER` clause. The `DEFINER` value specifies the security context to be used when checking access privileges at routine invocation time if the routine has the `SQL SECURITY DEFINER` characteristic. See [Section 17.2.1, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#), for more information.

When **mysqldump** is invoked with the `--routines` option, it now dumps the `DEFINER` value for stored routines.

- Large file support added to build for QNX platform. (Bug #17336)
- Large file support was re-enabled for the MySQL server binary for the AIX 5.2 platform. (Bug #13571)

Bugs fixed:

- If the `WHERE` condition of a query contained an OR-ed `FALSE` term, the set of tables whose rows cannot serve for null-complements in outer joins was determined incorrectly. This resulted in blocking possible conversions of outer joins into joins by the optimizer for such queries. (Bug #17164)
- **mysql_config** returned incorrect libraries on `x86_64` systems. (Bug #13158)

- Stored routine names longer than 64 characters were silently truncated. Now the limit is properly enforced and an error occurs. (Bug #17015)
- During conversion from one character set to ucs2, multi-byte characters with no ucs2 equivalent were converted to multiple characters, rather than to 0x003F QUESTION MARK. (Bug #15375)
- The `mysql_close()` C API function leaked handles for shared-memory connections on Windows. (Bug #15846)
- Checks for permissions on database operations could be performed in a case-insensitive manner (a user with permissions on database MYDATABASE could by accident get permissions on database myDataBase), if the privilege data were still cached from a previous check. (Bug #17279)
- If InnoDB ran out of buffer space for row locks and adaptive hashes, the server would crash. Now InnoDB rolls back the transaction. (Bug #18238)
- InnoDB tables with an adaptive hash blocked other queries during CHECK TABLE statements while the entire hash was checked. This could be a long time for a large hash. (Bug #17126)
- For InnoDB tables created in MySQL 4.1 or earlier, or created in 5.0 or later with compact format, updating a row so that a long column is updated or the length of some column changes, InnoDB later would fail to reclaim the BLOB storage space if the row was deleted. (Bug #18252)
- InnoDB had a memory leak for duplicate-key errors with tables having 90 columns or more. (Bug #18384)
- InnoDB: The LATEST FOREIGN KEY ERROR section in the output of SHOW INNODB STATUS was sometimes formatted incorrectly, causing problems with scripts that parsed the output of this statement. (Bug #16814)
- When using ORDER BY with a non-string column inside GROUP_CONCAT() the result's character set was converted to binary. (Bug #18281)

See also Bug #14169.

- `SELECT ... WHERE column LIKE 'A%'` when `column` had a key and used

the latin2_czech_cs collation. (Bug #17374)

- Complex queries with nested joins could cause a server crash. (Bug #18279)
- The server could deadlock under heavy load while writing to the binary log. (Bug #18116)
- A `SELECT ... ORDER BY ...` from a view defined using a function could crash the server. An example of such a view might be `CREATE VIEW AS SELECT SQRT(c1) FROM t1`. (Bug #18386)
- A `DELETE` using a subquery could crash the server. (Bug #18306)
- `REPAIR TABLE`, `OPTIMIZE TABLE`, and `ALTER TABLE` operations on transactional tables (or on tables of any type on Windows) could corrupt triggers associated with those tables. (Bug #18153)
- MyISAM: Performing a bulk insert on a table referenced by a trigger would crash the table. (Bug #17764)
- MyISAM: Keys for which the first part of the key was a `CHAR` or `VARCHAR` column using the UTF-8 character set and longer than 254 bytes could become corrupted. (Bug #17705)
- Using `ORDER BY intvar` within a stored procedure (where *intvar* is an integer variable or expression) would crash the server. (Bug #16474)

Note: The use of an integer *i* in an `ORDER BY i` clause for sorting the result by the *i*th column is deprecated (and non-standard). It should *not* be used in new applications. See [Section 13.2.7, “SELECT Syntax”](#).

- Triggers created in MySQL 5.0.16 and earlier could not be dropped after upgrading the server to 5.0.17 or later. (Bug #15921)
- A `SELECT` using a function against a nested view would crash the server. (Bug #15683)
- NDB cluster: Certain queries using `ORDER BY ... ASC` in the `WHERE` clause could return incorrect results. (Bug #17729)

- NDB Cluster: A timeout in the handling of an ABORT condition with more than 32 operations could yield a node failure. (Bug #18414)
- NDB Cluster: A node restart immediately following a CREATE TABLE would fail. **Important:** This fix supports 2-node Clusters only. (Bug #18385)
- NDB Cluster: In event of a node failure during a rollback, a “false” lock could be established on the backup for that node, which lock could not be removed without restarting the node. (Bug #18352)
- NDB Cluster: The cluster created a crashed replica of a table having an ordered index — or when logging was not enabled, of a table having a table or unique index — leading to a crash of the cluster following 8 successive restarts. (Bug #18298)
- NDB Cluster: When replacing a failed master node, the replacement node could cause the cluster to crash from a buffer overflow if it had an excessively large amount of data to write to the cluster log. (Bug #18118)
- NDB Cluster: If a **mysql** or other client could not parse the result set returned from a **mysqld** process acting as an SQL node in a cluster, the client would crash instead of returning the appropriate error. For example, this could happen when the client attempted to use a character set was not available to the **mysqld**. (Bug #17380)
- NDB Cluster: Restarting nodes were allowed to start and join the cluster too early. (Bug #16772)
- If a row was inserted inside a stored procedure using the parameters passed to the procedure in the INSERT statement, the resulting binlog entry was not escaped properly. (Bug #18293)
- If InnoDB encountered a HA_ERR_LOCK_TABLE_FULL error and rolled-back a transaction, the transaction was still written to the binary log. (Bug #18283)
- Stored procedures that call UDFs and pass local string variables caused server crashes. (Bug #17261)
- Connecting to a server with a UCS2 default character set with a client using a non-UCS2 character set crashed the server. (Bug #18004)

- Loading of UDFs in a statically linked MySQL caused a server crash. UDF loading is now blocked if the MySQL server is statically linked. (Bug #11835)
- Views that incorporate tables from the INFORMATION_SCHEMA resulted in a server crash when queried. (Bug #18224)
- A `SELECT *` query on an INFORMATION_SCHEMA table by a user with limited privileges resulted in a server crash. (Bug #18113)
- Attempting to access an InnoDB table after starting the server with `--skip-innodb` caused a server crash. (Bug #14575)
- InnoDB used table locks (not row locks) within stored functions. (Bug #18077)
- Replication slaves could not replicate triggers from older servers that included no `DEFINER` clause in the trigger definition. Now the trigger executes with the privileges of the invoker (which on the slave is the slave SQL thread). (Bug #16266)
- Character set conversion of string constants for `UNION` of constant and table column was not done when it was safe to do so. (Bug #15949)
- The `DEFINER` value for stored routines was not replicated. (Bug #15963)
- Use of stored functions with `DISTINCT` or `GROUP BY` can produce incorrect results when `ORDER BY` is also used. (Bug #13575)
- Use of `TRUNCATE TABLE` for a `TEMPORARY` table on a master server was propagated to slaves properly, but slaves did not decrement the `Slave_open_temp_tables` counter properly. (Bug #17137)
- `SELECT COUNT(*)` for a `MyISAM` table could return different results depending on whether an index was used. (Bug #14980)
- A `LEFT JOIN` with a `UNION` that selects literal values could crash the server. (Bug #17366)
- Large file support did not work in AIX server binaries. (Bug #10776)

- Updating a view that filters certain rows to set a filtered out row to be included in the table caused infinite loop. For example, if the view has a WHERE clause of salary > 100 then issuing an UPDATE statement of SET salary = 200 WHERE id = 10, caused an infinite loop. (Bug #17726)
- Certain combinations of joins with mixed ON and USING clauses caused unknown column errors. (Bug #15229)
- NDB Cluster: Inserting and deleting BLOB column values while a backup was in process could cause the loss of an **ndbd** node. (Bug #14028)
- If the server was started with the --skip-grant-tables option, it was impossible to create a trigger or a view without explicitly specifying a DEFINER clause. (Bug #16777)
- COUNT(DISTINCT col1, col2) and COUNT(DISTINCT CONCAT(col1, col2)) operations produced different results if one of the columns was an indexed DECIMAL column. (Bug #15745)
- The server displayed garbage in the error message warning about bad assignments to DECIMAL columns or routine variables. (Bug #15480)
- The server would execute stored routines that had a non-existent definer. (Bug #13198)
- For FEDERATED tables, a SELECT statement with an ORDER BY clause did not return rows in the proper order. (Bug #17377)
- The FORMAT() function returned an incorrect result when the client's character_set_connection value was utf8. (Bug #16678)
- NDB Cluster: Some query cache statistics were not always correctly reported for Cluster tables. (Bug #16795)
- Updating the value of a Unicode VARCHAR column with the result returned by a stored function would cause the insertion of ASCII characters into the column instead of Unicode, even where the function's return type was also declared as Unicode. (Bug #17615)

D.1.8. Changes in release 5.0.19 (04 March 2006)

Functionality added or changed:

- **Incompatible change:** The InnoDB storage engine no longer ignores trailing spaces when comparing BINARY or VARBINARY column values. This means that (for example) the binary values 'a' and 'a ' are now regarded as *unequal* any time they are compared, as they are in MyISAM tables. (Bug #14189)

See [Section 11.4.2, “The BINARY and VARBINARY Types”](#) for more information about the BINARY and VARBINARY types.

- Several changes were made to make upgrades easier:
 - Added the **mysql_upgrade** program that checks all tables for incompatibilities with the current version of MySQL Server and repairs them if necessary. This program should be run for each MySQL upgrade (rather than **mysql_fix_privilege_tables**). See [Section 5.6.2, “mysql_upgrade — Check Tables for MySQL Upgrade”](#).
 - Added the FOR UPGRADE option for the CHECK TABLE statement. This option checks whether tables are incompatible with the current version of MySQL Server.
 - Added the --check-upgrade to **mysqlcheck** that invokes CHECK TABLE with the FOR UPGRADE option.
- NDB Cluster: The **ndb_mgm** client commands node_id START and node_id STOP now work with management nodes as well as data nodes. (However, using ALL for the node_id continues to affect all data nodes only.)
- When using the GROUP_CONCAT() function where the group_concat_max_len system variable was greater than 512, the type of the result was BLOB only if the query included an ORDER BY clause; otherwise the result was a VARCHAR.

The result type of the GROUP_CONCAT() function is now VARCHAR only if the value of the group_concat_max_len system variable is less than or equal to 512. Otherwise, this function returns a BLOB. (Bug #14169)

- **mysql** no longer terminates data value display when it encounters a NUL byte. Instead, it displays NUL bytes as spaces. (Bug #16859)
- Added the `--wait-timeout` option to **mysqlmanager** to allow configuration of the timeout for dropping an inactive connection, and increased the default timeout from 30 seconds to 28,800 seconds (8 hours). (Bug #12674, Bug#15980)
- A number of performance issues were resolved that had previously been encountered when using statements that repeatedly invoked stored functions. For example, calling `BENCHMARK()` using a stored function executed much more slowly than when invoking it with inline code that accomplished the same task. In most cases the two should now execute with approximately the same speed. (Bug #15014, Bug #14946)
- `libmysqlclient` now uses versioned symbols with GNU ld. (Bug #3074)
- **NDB Cluster**: More descriptive warnings are now issued when inappropriate logging parameters are set in `config.ini`. (Formerly, the warning issued was simply Could not add logfile destination.) (Bug #11331)
- Added the `--port-open-timeout` option to **mysqld** to control how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. (Bug #15591)
- Repeated invocation of `my_init()` and `my_end()` caused corruption of character set data and connection failure. (Bug #6536)
- Two new Hungarian collations are included: `utf8_hungarian_ci` and `ucs2_hungarian_ci`. These support the correct sort order for Hungarian vowels. However, they do not support the correct order for sorting Hungarian consonant contractions; this issue will be fixed in a future release.
- Wording of error 1329 changed to No data - zero rows fetched, selected, or processed. (Bug #15206)
- The `INFORMATION_SCHEMA` now skips data contained in unlistable/unreadable directories rather than returning an error. (Bug

#15851)

- InnoDB now caches a list of unflushed files instead of scanning for unflushed files during a table flush operation. This improves performance when `--innodb-file-per-table` is set on a system with a large number of InnoDB tables. (Bug #15653)
- The message for error 1109 changed from Unknown table ... in order clause to Unknown table ... in field list. (Bug #15091)
- The `mysqltest` utility now converts all CR/LF combinations to LF to allow test cases intended for Windows to work properly on UNIX-like systems. (Bug #13809)
- The `mysql_ping` function will now retry if the reconnect flag is set and error CR_SERVER_LOST is encountered during the first attempt to ping the server. (Bug #14057)
- `mysqldump` now surrounds the `DEFINER`, `SQL SECURITY DEFINER` and `WITH CHECK OPTION` clauses of a `CREATE VIEW` statement with "not in version" comments to prevent errors in earlier versions of MySQL. (Bug #14871)
- New `charset` command added to **mysql** command-line client. By typing `charset name` or `\C name` (such as `\C UTF8`), the client character set can be changed without reconnecting. (Bug #16217)
- Client API will now attempt reconnect on TCP/IP if the reconnect flag is set, as is the case with sockets. (Bug #2845)

Bugs fixed:

- Generating an `AUTO_INCREMENT` value through a `FEDERATED` table did not set the value returned by `LAST_INSERT_ID()`. (Bug #14768)
- Cursors in stored routines could cause a server crash. (Bug #16887)
- Setting the `myisam_repair_threads` system variable to a value larger than 1 could cause corruption of large MyISAM tables. (Bug #11527)
- The length of a `VARCHAR()` column that used the `utf8` character set would

increase each time the table was re-created in a stored procedure or prepared statement, eventually causing the `CREATE TABLE` statement to fail. (Bug #13134)

- `type_decimal` failed with the prepared statement protocol. (Bug #17826)
- The MySQL server could crash with out of memory errors when performing aggregate functions on a `DECIMAL` column. (Bug #17602)
- A stored procedure failed to return data the first time it was called per connection. (Bug #17476)
- Using `DROP FUNCTION IF EXISTS func_name` to drop a user-defined function caused a server crash if the server was running with the `--skip-grant-tables` option. (Bug #17595)
- Using `ALTER TABLE` to increase the length of a `BINARY(M)` column caused column values to be padded with spaces rather than `0x00` bytes. (Bug #16857)
- A large `BIGINT` value specified in a `WHERE` clause could be treated differently depending on whether it is specified as a quoted string. (For example, `WHERE bigint_col = 17666000000000000000` versus `WHERE bigint_col = '17666000000000000000'`). (Bug #9088)
- A natural join between `INFORMATION_SCHEMA` tables failed. (Bug #17523)
- A memory leak caused warnings on slaves for certain statements that executed without warning on the master. (Bug #16175)
- The embedded server did not allow binding of columns to the `MYSQL_TYPE_VAR_STRING` data type in prepared statements. (Bug #12070)
- The embedded server failed various tests in the automated test suite. (Bug #9630, Bug #9631, Bug #9633, Bug #10801, Bug #10911, Bug #10924, Bug #10925, Bug #10926, Bug #10930, Bug #15433)
- Instance Manager erroneously accepted a list of instance identifiers for the `START INSTANCE` and `STOP INSTANCE` commands (should accept only a single identifier). (Bug #12813)

- For a transaction that used MyISAM and InnoDB tables, interruption of the transaction due to a dropped connection on a master server caused slaves to lose synchrony. (Bug #16559)
- SELECT with GROUP BY on a view can cause a server crash. (Bug #16382)
- If the query optimizer transformed a GROUP BY clause in a subquery, it did not also transform the HAVING clause if there was one, producing incorrect results. (Bug #16603)
- SUBSTRING_INDEX() could yield inconsistent results when applied with the same arguments to consecutive rows in a query. (Bug #14676)
- The parser allowed CREATE AGGREGATE FUNCTION for creating stored functions, even though AGGREGATE does not apply. (It is used only for CREATE FUNCTION only when creating user-defined functions.) (Bug #16896)
- Data truncations on non-UNIQUE indexes could crash InnoDB when using multi-byte character sets. (Bug #17530)
- Triggers created without BEGIN and END clauses could not be properly restored from a mysqldump file. (Bug #16878)
- The RENAME TABLE statement did not move triggers to the new table. (Bug #13525)
- Clients compiled from source with the --without-readline did not save command history from session to session. (Bug #16557)
- Stored routines that contained only a single statement were not written properly to the dumpfile when using mysqldump. (Bug #14857)
- For certain MERGE tables, the optimizer wrongly assumed that using index_merge/intersection was too expensive. (Bug #17314)
- Executing a SHOW CREATE VIEW query of an invalid view caused the mysql_next_result function of libMySQL.dll to hang. (Bug #15943)
- BIT fields were not properly handled when using row-based replication.

(Bug #13418)

- Issuing `GRANT EXECUTE` on a procedure would display any warnings related to the creation of the procedure. (Bug #7787)
- NDB Cluster: **`ndb_delete_all`** would run out of memory on tables containing BLOB columns. (Bug #16693)
- NDB Cluster: UNIQUE keys in Cluster tables were limited to 225 bytes in length. (Bug #15918)
- In a highly concurrent environment, a server crash or deadlock could result from execution of a statement that used stored functions or activated triggers coincident with alteration of the tables used by these functions or triggers. (Bug #16593)
- Previously, a stored function invocation was written to the binary log as `DO func_name()` if the invocation changes data and occurs within a non-logged statement, or if the function invokes a stored procedure that produces an error. These invocations now are logged as `SELECT func_name()` instead for better control over error code checking (slave servers could stop due to detecting a different error than occurred on the master). (Bug #14769)
- `CHECKSUM TABLE` returned different values on MyISAM table depending on whether the `QUICK` or `EXTENDED` options were used. (Bug #8841)
- MySQL server dropped client connection for certain `SELECT` statements against views defined that used `MERGE` algorithm. (Bug #16260)
- A call to the `IF()` function using decimal arguments could return incorrect results. (Bug #16272)
- A statement containing `GROUP BY` and `HAVING` clauses could return incorrect results when the `HAVING` clause contained logic that returned `FALSE` for every row. (Bug #14927)
- Using `GROUP BY` on column used in `WHERE` clause could cause empty set to be returned. (Bug #16203)
- For a MySQL 5.0 server, using MySQL 4.1 tables in queries with a `GROUP`

BY clause could result in buffer overrun or a server crash. (Bug #16752)

- SET sql_mode = N, where $N > 31$, did not work properly. (Bug #13897)
- NDB Cluster: Cluster log file paths were truncated to 128 characters. They may now be as long as MAX_PATH (the maximum path length permitted by the operating system). (Bug #17411)
- The mysql_stmt_store_result() C API function could not be used for a prepared statement if a cursor had been opened for the statement. (Bug #14013)
- The mysql_stmt_sqlstate() C API function incorrectly returned an empty string rather than '00000' when no error occurred. (Bug #16143)
- Using the TRUNCATE() function with a negative number for the second argument on a BIGINT column returned incorrect results. (Bug #8461)
- Instance Manager searched wrong location for password file on some platforms. (Bug #16499)
- NDB Cluster: Following multiple forced shutdowns and restarts of data nodes, DROP DATABASE could fail. (Bug #17325)
- NDB Cluster: An UPDATE with an inner join failed to match any records if both tables in the join did not have a primary key. (Bug #17257)
- NDB Cluster: A DELETE with a join in the WHERE clause failed to retrieve any records if both tables in the join did not have a primary key. (Bug #17249)
- The error message returned by perror --ndb was prefixed with OS error code: instead of NDB error code:. (Bug #17235)
- NDB Cluster: In some cases, LOAD DATA INFILE did not load all data into NDB tables. (Bug #17081)
- NDB Cluster: The REDO log would become corrupted (and thus unreadable) in some circumstances, due to a failure in the query handler. (Bug #17295)

- NDB Cluster: No error message was generated for setting `NoOfFragmentLogFiles` too low. (Bug #13966)
- NDB Cluster: No error message was generated for setting `MaxNoOfAttributes` too low. (Bug #13965)
- Binary distributions for Solaris contained files with group ownership set to the non-existing `wheel` group. Now the `bin` group is used. (Bug #15562)
- The `DECIMAL` data type was not being handled correctly with prepared statements. (Bug #16511)
- The `SELECT` privilege was required for triggers that performed no selects. (Bug #15196)
- The `UPDATE` privilege was required for triggers that performed no updates. (Bug #15166)
- `CAST(... AS TIME)` operations returned different results when using versus not using prepared-statement protocol. (Bug #15805)
- Improper memory handling for stored routine variables could cause memory overruns and binary log corruption. (Bug #15588)
- Killing a long-running query containing a subquery could cause a server crash. (Bug #14851)
- A `FULLTEXT` query in a prepared statement could result in unexpected behavior. (Bug #14496)
- A `RETURN` statement within a trigger caused a server crash. `RETURN` now is disallowed within triggers. To exit immediately, use `LEAVE`. (Bug #16829)
- `STR_TO_DATE(1, NULL)` caused a server crash. ([CVE-2006-3081](#), Bug #15828)
- An invalid stored routine could not be dropped. (Bug #16303)
- When evaluation of the test in a `CASE` failed in a stored procedure that contained a `CONTINUE` handler, execution resumed at the beginning of the

CASE statement instead of at the end. (Bug #16568)

- An INSERT statement in a stored procedure corrupted the binary log. (Bug #16621)
- When MyODBC or any other client called `my_init()/my_end()` several times, it caused corruption of charset data stored in `once_mem_pool`. (Bug #11892)
- When multiple handlers are created for the same MySQL error number within nested blocks, the outermost handler took precedence. (Bug #15011)
- Certain LEAVE statements in stored procedures were not properly optimized. (Bug #15737)
- Setting InnoDB path settings to an empty string caused InnoDB storage engine to crash upon server startup. (Bug #16157)
- InnoDB used full explicit table locks in trigger processing. (Bug #16229)
- Server crash when dropping InnoDB constraints named `TABLENAME_ibfk_0`. (Bug #16387)
- Corrected race condition when dropping the adaptive hash index for a B-tree page in InnoDB. (Bug #16582)
- The `mysql_real_connect()` C API function incorrectly reset the `MYSQL_OPT_RECONNECT` option to its default value. (Bug #15719)
- InnoDB: After upgrading an InnoDB table having a `VARCHAR BINARY` column created in MySQL 4.0 to MySQL 5.0, update operations on the table would cause the server to crash. (Bug #16298)
- Trying to compile the server on Windows generated a stack overflow warning due to a recursive definition of the internal `Field_date::store()` method. (Bug #15634)
- The use of `LOAD INDEX` within a stored routine was permitted and caused the server to crash. **Note:** `LOAD INDEX` statements within stored routines *are not supported*, and now yield an error if attempted. This behavior is

intended. (Bug #14270)

- The **mysqlbinlog** utility did not output DELIMITER statements, causing syntax errors for stored routine creation statements. (Bug #11312)
- NDB Cluster returned incorrect Can't find file error for OS error 24, changed to Too many open files. (Bug #15020)
- Performing a RENAME TABLE on an InnoDB table when the server is started with the --innodb-file-per-table and the data directory is a symlink caused a server crash. (Bug #15991)
- Multi-byte path names for LOAD DATA and SELECT ... INTO OUTFILE caused errors. Added the character_set_filesystem system variable, which controls the interpretation of string literals that refer to filenames. (Bug #12448)
- Certain subqueries where the inner query is the result of a aggregate function would return different results on MySQL 5.0 than on MySQL 4.1. (Bug #15347)
- Attempts to create FULLTEXT indexes on VARCHAR columns larger than 1000 bytes resulted in error. (Bug #13835)
- Characters in the gb2312 and euckr character sets which did not have Unicode mappings were truncated. (Bug #15377)
- Certain nested LEFT JOIN operations were not properly optimized. (Bug #16393)
- GRANT statements specifying schema names that included underscore characters (i.e. my_schema) did not match if the underscore was escaped in the GRANT statement (i.e. GRANT ALL ON `my_schema` ...). (Bug #14834)
- Running out of disk space in the location specified by the tmpdir option resulted in incorrect error message. (Bug #14634)
- Test suite sp test left behind tables when the test failed that could cause future tests to fail. (Bug #15866)

- UPDATE statement crashed multi-byte character set FULLTEXT index if update value was almost identical to initial value only differing in some spaces being changed to . (Bug #16489)
- A SELECT query which contained a GROUP_CONCAT() and an ORDER BY clause against the INFORMATION_SCHEMA resulted in an empty result set. (Bug #15307)
- The --replicate-do and --replicate-ignore options were not being enforced on multiple-table statements. (Bug #15699, Bug #16487)
- A prepared statement created from a SELECT ... LIKE query (such as PREPARE stmt1 FROM 'SELECT col_1 FROM tedd_test WHERE col_1 LIKE ?';) would begin to produce erratic results after being executed repeatedly numerous (thousands) of times. (Bug #12734)
- The server would crash when the size of an ARCHIVE table grew beyond 2GB. (Bug #15787)
- Created a user function with an empty string (that is, CREATE FUNCTION ''()), was accepted by the server. Following this, calling SHOW FUNCTION STATUS would cause the server to crash. (Bug #15658)
- In some cases the query optimizer did not properly perform multiple joins where inner joins followed left joins, resulting in corrupted result sets. (Bug #15633)
- The absence of a table in the left part of a left or right join was not checked prior to name resolution, which resulted in a server crash. (Bug #15538)
- NDBCluster: A bitfield whose offset and length totaled 32 would crash the cluster. (Bug #16125)
- NDBCluster: Upon the completion of a scan where a key request remained outstanding on the primary replica and a starting node died, the scan did not terminate. This caused incompleting error handling of the failed node. (Bug #15908)
- NDBCluster: The ndb_autodiscover test failed sporadically due to a node not being permitted to connect to the cluster. (Bug #15619)

- NDBCluster: When running more than one management process in a cluster:
 - **ndb_mgm -c host:port -e "node_id stop"** would stop a management process running only on the same system on which the command was issued.
 - **ndb_mgm -e "shutdown"** failed to shut down any management processes at all.

(Bug #12045, Bug #12124)

- The contents of `fill_help_tables.sql` could not be loaded in strict SQL mode. (Bug #15760)
- `fill_help_tables.sql` was not included in binary distributions for several platforms. (Bug #15759)
- An `INSERT ... SELECT` statement between tables in a MERGE set can return errors when statement involves insert into child table from merge table or vice-versa. (Bug #5390)
- Certain permission management statements could create a NULL hostname for a user, resulting in a server crash. (Bug #15598)
- A `COMMIT` statement followed by a `ALTER TABLE` statement on a BDB table caused server crash. (Bug #14212)
- A `DELETE` statement involving a `LEFT JOIN` and an `IS NULL` test on the right-hand table of the join crashed the server when the `innodb_locks_unsafe_for_binlog` option was enabled. (Bug #15650)
- Performing an `ORDER BY` on an indexed `ENUM` column returned error. (Bug #15308)
- The `NOT FOUND` condition handler for stored procedures did not distinguish between a `NOT FOUND` condition and an exception or warning. (Bug #15231)
- A stored procedure with an undefined variable and an exception handler would hang the client when called. (Bug #14498)

- Subselect could return wrong results when records cache and grouping was involved. (Bug #15347)
- Temporary table aliasing did not work inside stored functions. (Bug #12198)
- MIN() and MAX() operations were not optimized for views. (Bug #16016)
- Using an aggregate function as the argument for a HAVING clause would result in the aggregate function always returning FALSE. (Bug #14274)
- Parallel builds occasionally failed on Solaris. (Bug #16282)
- The FORCE INDEX keyword in a query would prevent an index merge from being used where an index merge would normally be chosen by the optimizer. (Bug #16166)
- The COALESCE() function truncated data in a TINYTEXT column. (Bug #15581)
- InnoDB: Comparison of indexed VARCHAR CHARACTER SET ucs2 COLLATE ucs2_bin columns using LIKE could fail. (Bug #14583)
- An attempt to open a table that requires a disabled storage engine could cause a server crash. (Bug #15185)
- Issuing a DROP USER command could cause some users to encounter a hostname is not allowed to connect to this MySQL server error. (Bug #15775)
- Setting innodb_log_file_size to a value greater than 4G crashed the server. (Bug #15108)
- A SELECT of a stored function that references the INFORMATION_SCHEMA could crash the server. (Bug #15533)
- Tarball install package was missing a proper fill_help_tables.sql file. (Bug #15151)

D.1.9. Changes in release 5.0.18 (21 December 2005)

Functionality added or changed:

- It is now possible to build the server such that MyISAM tables can support up to 128 keys rather than the standard 64. This can be done by configuring the build using the option `--with-max-indexes=N`, where $N \leq 128$ is the maximum number of indexes to permit per table. (Bug #10932)
- The server treats stored routine parameters and local variables (and stored function return values) according to standard SQL. Previously, parameters, variables, and return values were treated as items in expressions and were subject to automatic (silent) conversion and truncation. Now the data type is observed. Data type conversion and overflow problems that occur in assignments result in warnings, or errors in strict mode. The `CHARACTER SET` clause for character data type declarations is used. Parameters, variables, and return values must be scalars; it is no longer possible to assign a row value. Also, stored functions execute using the `sql_mode` value in force at function creation time rather than ignoring it. For more information, see [Section 17.2.1, “CREATE PROCEDURE and CREATE FUNCTION Syntax”](#). (Bug #8702, Bug #8768, Bug #8769, Bug #9078, Bug #9572, Bug #12903, Bug #13705, Bug #13808, Bug #13909, Bug #14161, Bug #15148)

Bugs fixed:

- API function `mysql_stmt_prepare` returned wrong field length for TEXT columns. (Bug #15613)
- The output of `mysqldump --triggers` did not contain the `DEFINER` clause in dumped trigger definitions. (Bug #15110)
- The output of `SHOW TRIGGERS` contained extraneous whitespace. (Bug #15103)
- Creating a trigger caused a server crash if the table or trigger database was not known because no default database had been selected. (Bug #14863)
- `SHOW [FULL] COLUMNS` and `SHOW INDEX FROM` did not function with temporary tables. (Bug #14271, Bug #14387, Bug #15224)
- The `INFORMATION_SCHEMA.COLUMNS` table did not report the size of `BINARY` or `VARBINARY` columns. (Bug #14271)

- The server would not compile under Cygwin. (Bug #13640)
- DESCRIBE did not function with temporary tables. (Bug #12770)
- Reversing the order of operands in a WHERE clause testing a simple equality (such as WHERE t1.col1 = t2.col2) would produce different output from EXPLAIN. (Bug #15106)
- Column aliases were displayed incorrectly in a SELECT from a view following an update to a base table of the view. (Bug #14861)
- Set functions could not be aggregated in outer subqueries. (Bug #12762)
- When a connection using yaSSL was aborted, the server would continue to try to read the closed socket, and the thread continued to appear in the output of SHOW PROCESSLIST. Note that this issue did not affect secure connection attempts using OpenSSL. (Bug #15772)
- InnoDB: Having two tables in a parent-child relationship enforced by a foreign key where one table used ROW_FORMAT=COMPACT and the other used ROW_FORMAT=REDUNDANT could result in a MySQL server crash. Note that this problem did not exist prior to MySQL 5.0.3, when the compact row format for InnoDB was introduced. (Bug #15550)
- BDB: A DELETE, INSERT, or UPDATE of a BDB table could cause the server to crash where the query contained a subquery using an index read. (Bug #15536)
- A left join on a column that having a NULL value could cause the server to crash. (Bug #15268)
- A replication slave server could sometimes crash on a BEFORE UPDATE trigger if the UPDATE query was not executed in the same database as the table with the trigger. (Bug #14614)
- A race condition when creating temporary files caused a deadlock on Windows with threads in opening tables or waiting for table states. (Bug #12071)
- InnoDB: If FOREIGN_KEY_CHECKS was 0, InnoDB allowed inconsistent foreign

keys to be created. (Bug #13778)

- **NDB cluster:** Under some circumstances, it was possible for a restarting node to undergo a forced shutdown. (Bug #15632)
- **NDB cluster:** If an abort by the Transaction Coordinator timed out, the abort condition was incorrectly handled, causing the transaction record to be released prematurely. (Bug #15685)
- **NDB cluster:** The `ndb_read_multi_range.test` script failed to drop a table, causing the test to fail. (Bug #15675) (See also Bug #15401.)
- **NDB cluster:** A node which failed during cluster startup was sometimes not removed from the internal list of active nodes. (Bug #15587)
- Resolution of the argument to the `VALUES()` function to a variable inside a stored routine caused a server crash. The argument must be a table column. (Bug #15441)

D.1.10. Changes in release 5.0.17 (14 December 2005)

Functionality added or changed:

- The original Linux RPM packages (5.0.17-0) had an issue with a `zlib` dependency that would result in an error during an install or upgrade. They were replaced by new binaries, 5.0.17-1. (Bug #15223) Here is a list of the new RPM binaries:
 - `MySQL-{Max,client,devel,server,shared,ndb*}-5.0.17-1.i386.rpm`
 - `MySQL-*-standard-5.0.17-1.rhel3.i386.rpm`, `MySQL-*-standard-5.0.17-1.rhel3.ia64.rpm`, `MySQL-*-standard-5.0.17-1.rhel3.x86_64.rpm`
 - `MySQL-*-pro-5.0.17-1.rhel3.i386.rpm`, `MySQL-*-pro-5.0.17-1.rhel3.ia64.rpm`, `MySQL-*-pro-5.0.17-1.rhel3.x86_64.rpm`
 - `MySQL-*-pro-gpl-5.0.17-1.rhel3.i386.rpm`, `MySQL-*-pro-gpl-5.0.17-1.rhel3.ia64.rpm`, `MySQL-*-pro-gpl-5.0.17-1.rhel3.x86_64.rpm`

- The syntax for `CREATE TRIGGER` now includes a `DEFINER` clause for specifying which access privileges to check at trigger invocation time. See [Section 18.1, “CREATE TRIGGER Syntax”](#), for more information.

Known issue: If you attempt to replicate from a master server older than MySQL 5.0.17 to a slave running MySQL 5.0.17 through 5.0.19, replication of `CREATE TRIGGER` statements fails on the slave with a `Definer not fully qualified` error. A workaround is to create triggers on the master using a version-specific comment embedded in each `CREATE TRIGGER` statement:

```
CREATE /*!50017 DEFINER = 'root'@'localhost' */ TRIGGER ... ;
```

`CREATE TRIGGER` statements written this way will replicate to newer slaves, which pick up the `DEFINER` clause from the comment and execute successfully. (Bug #16266)

- Added a `DEFINER` column to the `INFORMATION_SCHEMA.TRIGGERS` table.
- Invoking a stored function or trigger creates a new savepoint level. When the function or trigger finishes, the previous savepoint level is restored. (See Bug #13825 for more information.)
- Recursion is allowed in stored procedures. Recursive stored functions and triggers still are disallowed. (Bug #10100)
- In the `latin5_turkish_ci` collation, the order of the characters `A WITH CIRCUMFLEX`, `I WITH CIRCUMFLEX`, and `U WITH CIRCUMFLEX` was changed. If you have used these characters in any indexed columns, you should rebuild those indexes. (Bug #13421)
- Support files for compiling with Visual Studio 6 have been removed. (Bug #15094)

Bugs fixed:

- RPM packages had an incorrect `zlib` dependency. (Bug #15223)
- `NDB Cluster`: `REPLACE` failed when attempting to update a primary key value in a `Cluster` table. (Bug #14007)

- **make** failed when attempting to build MySQL in different directory than source. (Bug #11827)
- Corrected an error-handling problem within stored routines on 64-bit platforms. (Bug #15630)
- Slave SQL thread cleanup was not handled properly on Mac OS X when a statement was killed, resulting in a slave crash. (Bug #15623, Bug #15668)
- Symbolic links did not function properly on Windows platforms. (Bug #14960, Bug #14310)
- **mysqld** would not start on Windows 9X operating systems including Windows Me. (Bug #15209)
- InnoDB: During replication, There was a failure to record events in the binary log that still occurred even in the event of a ROLLBACK. For example, this sequence of commands:

```
BEGIN;
CREATE TEMPORARY TABLE t1 (a INT) ENGINE=INNODB;
ROLLBACK;
INSERT INTO t1 VALUES (1);
```

would succeed on the replication master as expected. However, the INSERT would fail on the slave because the ROLLBACK would (erroneously) cause the CREATE TEMPORARY TABLE statement not to be written to the binlog. (Bug #7947)

- A bug in `mysql-test/t/mysqltest.test` caused that test to fail. (Bug #15605)
- The CREATE test case in **mysql-test-run.pl** failed on AIX and SCO. (Bug #15607)
- NDB Cluster: Creating a table with packed keys failed silently. NDB now supports the PACK_KEYS option to CREATE TABLE correctly. (Bug #14514)
- NDB Cluster: Using ORDER BY primary_key_column when selecting from a table having the primary key on a VARCHAR column caused a forced shutdown of the cluster. (Bug #14828, Bug #15240, Bug #15682, Bug

#15517)

- **NDB cluster:** Under certain circumstances, when **mysqld** connects to a cluster management server, the connection would fail before a node ID could be allocated. (Bug #15215)
- **NDB cluster:** There was a small window for a node failure to occur during a backup without an error being reported. (Bug #15425)
- **mysql --help** was missing a newline after the version string when the bundled readline library was not used. (Bug #15097)
- Implicit versus explicit conversion of float to integer (such as inserting a float value into an integer column versus using `CAST(... AS UNSIGNED` before inserting the value) could produce different results. Implicit and explicit typecasts now are done the same way, with a value equal to the nearest integer according to the prevailing rounding mode. (Bug #12956)
- **GROUP BY** on a view column did not correctly account for the possibility that the column could contain `NULL` values. (Bug #14850)
- **ANALYZE TABLE** did not properly update table statistics for a MyISAM table with a `FULLTEXT` index containing stopwords, so a subsequent **ANALYZE TABLE** would not recognize the table as having already been analyzed. (Bug #14902)
- The maximum value of `MAX_ROWS` was handled incorrectly on 64-bit systems. (Bug #14155)
- **NDB cluster:** A forced cluster shutdown occurred when the management daemon was restarted with a changed `config.ini` file that added an `API/SQL` node. (Bug #15512)
- Multiple-table update operations were counting updates and not updated rows. As a result, if a row had several updates it was counted several times for the “rows matched” value but updated only once. (Bug #15028)
- A statement that produced a warning, when fetched via `mysql_stmt_fetch()`, did not produce a warning count according to `mysql_warning_count()`. (Bug #15510)

- Manual manipulation of the `mysql.proc` table could cause a server crash. This should not happen, but it is also not supported that the server will notice such changes. (Bug #14233)
- Revised table locking to allow proper assessment of view security. (Bug #11555)
- Within a stored procedure, inserting with `INSERT ... SELECT` into a table with an `AUTO_INCREMENT` column did not generate the correct sequence number. (Bug #14304)
- `SELECT` queries that began with an opening parenthesis were not being placed in the query cache. (Bug #14652)
- Space truncation was being ignored when inserting into `BINARY` or `VARBINARY` columns. Now space truncation results in a warning, or an error in strict mode. (Bug #14299)
- The database-changing code for stored routine handling caused an error-handling problem resulting in a server crash. (Bug #15392)
- Selecting from a view processed with the temptable algorithm caused a server crash if the query cache was enabled. (Bug #15119)
- `REPAIR TABLES`, `BACKUP TABLES`, `RESTORE TABLES` within a stored procedure caused a server crash. (Bug #13012)
- Creating a view that referenced a stored function that selected from a view caused a crash upon selection from the view. (Bug #15096)
- `ALTER TABLE ... SET DEFAULT` had no effect. (Bug #14693)
- Creating a view within a stored procedure could result in an out of memory error or a server crash. (Bug #14885)
- InnoDB: A race condition allowed two threads to drop a hash index simultaneously. (Bug #14747)
- **mysqlhotcopy** tried to copy `INFORMATION_SCHEMA` tables. (Bug #14610)

- `CHAR(... USING ...)` and `CONVERT(CHAR(...) USING ...)`, though logically equivalent, could produce different results. (Bug #14146)
- The value of `INFORMATION_SCHEMA.TABLES.TABLE_TYPE` sometimes was reported as empty. (Bug #14476)
- InnoDB: Activity on an InnoDB table caused execution time for `SHOW CREATE TABLE` for the table to increase. (Bug #13762)
- `DELETE` from CSV tables reported an incorrect rows-affected value. (Bug #13406)
- The server crashed if compiled without any transactional storage engines. (Bug #15047)
- Declaring a stored routine variable to have a `DEFAULT` value that referred to a variable of the same name caused a server crash. (For example: `DECLARE x INT DEFAULT x`) Now the `DEFAULT` variable is interpreted as referring to a variable in an outer scope, if there is one. (Bug #14376)
- Perform character set conversion of constant values whenever possible without data loss. (Bug #10446)
- **mysql** ignored the `MYSQL_TCP_PORT` environment variable. (Bug #5792)
- `ROW_COUNT()` returned an incorrect result after `EXECUTE` of a prepared statement. (Bug #14956)
- A `UNION` of `DECIMAL` columns could produce incorrect results. (Bug #14216)
- Queries that select records based on comparisons to a set of column could crash the server if there was one index covering the columns, and a set of other non-covering indexes that taken together cover the columns. (Bug #15204)
- When using an aggregate function to select from a table that has a multiple-column primary key, adding `ORDER BY` to the query could produce an incorrect result. (Bug #14920)
- `SHOW CREATE TABLE` for a view could fail if the client had locked the view.

(Bug #14726)

- For binary string data types, **mysqldump --hex-blob** produced an illegal output value of 0x rather than ' '. (Bug #13318)
- Some comparisons for the IN() operator were inconsistent with equivalent comparisons for the = operator. (Bug #12612)
- In a stored procedure, continuing (via a condition handler) after a failed variable initialization caused a server crash. (Bug #14643)
- Within a stored procedure, exception handling for UPDATE statements that caused a duplicate-key error caused a Packets out of order error for the following statement. (Bug #13729)
- Creating a table containing an ENUM or SET column from within a stored procedure or prepared statement caused a server crash later when executing the procedure or statement. (Bug #14410)
- Selecting from a view used filesort retrieval when faster retrieval was possible. (Bug #14816)
- Warnings from a previous command were not being reset when fetching from a cursor. (Bug #13524)
- RESET MASTER failed to delete log files on Windows. (Bug #13377)
- Using ORDER BY on a column from a view, when also selecting the column normally, and via an alias, caused a mistaken Column 'x' in order clause is ambiguous error. (Bug #14662)
- Invoking a stored procedure within another stored procedure caused the server to crash. (Bug #13549)
- Stored functions making use of cursors were not replicated. (Bug #14077)
- CAST(expr AS BINARY(*n*)) did not pad with 0x00 to a length of *n* bytes. (Bug #14255)
- Casting a FLOAT or DOUBLE whose value was less than 1.0E-06 to DECIMAL

would yield an inappropriate value. (Bug #14268)

- In some cases, a left outer join could yield an invalid result or cause the server to crash, due to a `MYSQL_DATA_TRUNCATED` error. (Bug #13488)
- For an invalid view definition, selecting from the `INFORMATION_SCHEMA.VIEWS` table or using `SHOW CREATE VIEW` failed, making it difficult to determine what part of the definition was invalid. Now the server returns the definition and issues a warning. (Bug #13818)
- The server could misinterpret old trigger definition files created before MySQL 5.0.17. Now they are interpreted correctly, but this takes more time and the server issues a warning that the trigger should be re-created. (Bug #14090)
- **mysqldump --triggers** did not account for the SQL mode and could dump trigger definitions with missing whitespace if the `IGNORE_SPACE` mode was enabled. (Bug #14554)
- Within a trigger definition the `CURRENT_USER()` function evaluated to the user whose actions caused the trigger to be activated. Now that triggers have a `DEFINER` value, `CURRENT_USER()` evaluates to the trigger definer. (Bug #5861)
- `CREATE TABLE tbl_name (...) SELECT ...` could crash the server and write invalid data into the `.frm` file if the `CREATE TABLE` and `SELECT` both contained a column with the same name. Also, if a default value is specified in the column definition, it is now actually used. (Bug #14480)
- A newline character in a column alias in a view definition caused an error when selecting from the view later. (Bug #13622)
- `mysql_fix_privilege_tables.sql` contained an erroneous comment that resulted in an error when the file contents were processed. (Bug #14469)
- On Windows, the server could crash during shutdown if both replication threads and normal client connection threads were active. (Re-fix of Bug #11796)
- The grammar for supporting the `DEFINER = CURRENT_USER` clause in

CREATE VIEW and ALTER VIEW was incorrect. (Bug #14719)

- Queries on ARCHIVE tables that used the filesort sorting method could result in a server crash. (Bug #14433)
- The `mysql_stmt_fetch()` C APP function could return `MYSQL_NO_DATA` for a `SELECT COUNT(*) FROM tbl_name WHERE 1 = 0` statement, which should return 1 row. (Bug #14845)
- A LIMIT-related optimization failed to take into account that MyISAM table indexes can be disabled, causing Error 124 when it tried to use such an index. (Bug #14616)
- A server crash resulted from the following sequence of events: 1) With no default database selected, create a stored procedure with the procedure name explicitly qualified with a database name (`CREATE PROCEDURE db_name.proc_name ...`). 2) Create another stored procedure with no database name qualifier. 3) Execute `SHOW PROCEDURE STATUS`. (Bug #14569)
- Complex subqueries could cause improper internal query execution environment initialization and crash the server. (Bug #14342)
- For a table that had been opened with `HANDLER OPEN`, issuing `OPTIMIZE TABLE`, `ALTER TABLE`, or `REPAIR TABLE` caused a server crash. (Bug #14397)
- A server crash could occur if a prepared statement invoked a stored procedure that existed when the statement was prepared but had been dropped and re-created prior to statement execution. (Bug #12329)
- A server crash could occur if a prepared statement updated a table for which a trigger existed when the statement was prepared but had been dropped prior to statement execution. (Bug #13399)
- Statements that implicitly commit a transaction are prohibited in stored functions and triggers. An attempt to create a function or trigger containing such a statement produces an error. (Bug #13627) (The originally reported symptom was that a trigger that dropped another trigger could cause a server crash. That problem was fixed by the patch for Bug #13343.)

D.1.11. Changes in release 5.0.16 (10 November 2005)

Functionality added or changed:

- When trying to run the server with yaSSL enabled, MySQL now tries to open `/dev/random` automatically if `/dev/urandom` is not available. (Bug #13164)
- The `read_only` system variable no longer applies to TEMPORARY tables. (Bug #4544)
- Due to changes in binary logging, the restrictions on which stored routine creators can be trusted not to create unsafe routines have been lifted for stored procedures (but not stored functions). Consequently, the `log_bin_trust_routine_creators` system variable and the corresponding `--log-bin-trust-routine-creators` server option were renamed to `log_bin_trust_function_creators` and `--log-bin-trust-function-creators`. For backward compatibility, the old names are recognized but result in a warning. See [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).
- Added the `Compression` status variable, which indicates whether the client connection uses compression in the client/server protocol.
- In MySQL 5.0.13, syntax for `DEFINER` and `SQL SECURITY` clauses was added to the `CREATE VIEW` and `ALTER VIEW` statements, but the clauses had no effect. They now are enabled. They specify the security context to be used when checking access privileges at view invocation time. See [Section 19.2, “CREATE VIEW Syntax”](#), for more information.
- The InnoDB, NDB, BDB, and ARCHIVE storage engines now support spatial columns. See [Chapter 16, *Spatial Extensions*](#).
- The `CHECK TABLE` statement now works for ARCHIVE tables.
- You must now declare a prefix for an index on any column of any Geometry class, the only exception being when the column is a `POINT`. (Bug #12267)
- Added a `--hexdump` option to **mysqlbinlog** that displays a hex dump of the log in comments. This output can be helpful for replication debugging.

- MySQL 5.0 now supports character set conversion for seven additional cp950 characters into the big5 character set: 0xF9D6, 0xF9D7, 0xF9D8, 0xF9D9, 0xF9DA, 0xF9DB, and 0xF9DC. **Note:** If you move data containing these additional characters to an older MySQL installation which does not support them, you may encounter errors. (Bug #12476)
- When a date column is set NOT NULL and contains 0000-00-00, it will be updated for UPDATE statements that contains columnname IS NULL in the WHERE clause. (Bug #14186)

Bugs fixed:

- When the DATE_FORMAT() function appeared in both the SELECT and ORDER BY clauses of a query but with arguments that differ by case (i.e. %m and %M), incorrect sorting may have occurred. (Bug #14016)
- For InnoDB tables, using a column prefix for a utf8 column in a primary key caused Cannot find record errors when attempting to locate records. (Bug #14056)
- NDB Cluster: A memory leak occurred when performing ordered index scans using indexes a columns larger than 32 bytes, which would eventually lead to the forced shutdown of all **mysqld** server processes used with the cluster. (Bug #13078)
- InnoDB: Large innobase_buffer_pool_size and innobase_log_file_size values were displayed incorrectly on 64-bit systems. (Bug #12701)
- InnoDB: When dropping and adding a PRIMARY KEY, if a loose index scan using only the second part of multiple-part index was chosen, incorrect keys were created and an endless loop resulted. (Bug #13293)
- NDB Cluster: Repeated transactions using unique index lookups could cause a memory leak leading to error 288, Out of index operations in transaction coordinator. (Bug #14199)
- Selecting from a table in both an outer query and a subquery could cause a server crash. (Bug #14482)
- SHOW CREATE TABLE did not display the CONNECTION string for FEDERATED

tables. (Bug #13724)

- For some stored functions dumped by **mysqldump --routines**, the function definition could not be reloaded later due to a parsing error. (Bug #14723)
- For a MyISAM table originally created in MySQL 4.1, `INSERT DELAYED` could cause a server crash. (Bug #13707)
- The `--exit-info=65536` option conflicted with `--temp-pool` and caused problems with the server's use of temporary files. Now `--temp-pool` is ignored if `--exit-info=65536` is specified. (Bug #9551)
- `ORDER BY DESC` within the `GROUP_CONCAT()` function was not honored when used in a view. (Bug #14466)
- A comparison with an invalid date (such as `WHERE col_name > '2005-09-31'`) caused any index on `col_name` not to be used and a string comparison for each row, resulting in slow performance. (Bug #14093)
- Within stored routines, `REPLACE()` could return an empty string (rather than the original string) when no replacement was done, and `IFNULL()` could return garbage results. (Bug #13941)
- Inserts of too-large `DECIMAL` values were handled inconsistently (sometimes set to the maximum `DECIMAL` value, sometimes set to 0). (Bug #13573)
- Executing `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE` on a view for which an underlying table had been dropped caused a server crash. (Bug #14540)
- A prepared statement that selected from a view processed using the merge algorithm could crash on the second execution. (Bug #14026)
- Deletes from a CSV table could cause table corruption. (Bug #14672)
- An update of a CSV table could cause a server crash. (Bug #13894)
- For queries with nested outer joins, the optimizer could choose join orders that query execution could not handle. The fix is that now the optimizer avoids choosing such join orders. (Bug #13126)

- Starting **mysqld** with the `--skip-innodb` and `--default-storage-engine=innodb` (or `--default-table-type=innodb`) caused a server crash. (Bug #9815, re-fix of bug from 5.0.5)
- **mysqlmanager** did not start up correctly on Windows 2003. (Bug #14537)
- The parser did not correctly recognize wildcards in the host part of the `DEFINER` user in `CREATE VIEW` statements. (Bug #14256)
- Memory corruption and a server crash could be caused by statements that used a cursor and generated a result set larger than `max_heap_table_size`. (Bug #14210)
- **mysqld_safe** did not correctly start the `-max` version of the server (if it was present) if the `--ledir` option was given. (Bug #13774)
- The **mysql** parser did not properly strip the delimiter from input lines less than nine characters long. For example, this could cause `USE abc;` to result in an `Unknown database: abc;` error. (Bug #14358)
- Statements of the form `CREATE TABLE ... SELECT ...` that created a column with a multi-byte character set could incorrectly calculate the maximum length of the column, resulting in a `Specified key was too long` error. (Bug #14139)
- Some updatable views could not be updated. (Bug #14027)
- Running `OPTIMIZE TABLE` and other data-updating statements concurrently on an `InnoDB` table could cause a crash or the following warnings in the error log: `Warning: Found locks from different threads in write: enter write_lock, Warning: Found locks from different threads in write: start of release lock`. (Bug #11704)
- Indexes for `BDB` tables were being limited incorrectly to 255 bytes. (Bug #14381)
- Use of `col_name = VALUES(col_name)` in the `ON DUPLICATE KEY UPDATE` clause of an `INSERT` statement failed with an `Column 'col_name' in field list is ambiguous` error. (Bug #13392)

- On Windows, the server was not ignoring hidden or system directories that Windows may have created in the data directory, and would treat them as available databases. (Bug #4375)
- **mysqldump** could not dump views if the `-x` option was given. (Bug #12838)
- **mysqlimport** now issues a `SET @@character_set_database = binary` statement before loading data so that a file containing mixed character sets (columns with different character sets) can be loaded properly. (Bug #12123)
- Use of the deprecated `--sql-bin-update-same` option caused a server crash. (Bug #12974)
- Maximum values were handled incorrectly for command-line options of type `GET_LL`. (Bug #12925)
- For a user that has the `SELECT` privilege on a view, the server erroneously was also requiring the user to have the `EXECUTE` privilege at view execution time for stored functions used in the view definition. (Bug #9505)
- Use of `WITH ROLLUP PROCEDURE ANALYSE()` could hang the server. (Bug #14138)
- `TIMEDIFF()`, `ADDTIME()`, and `STR_TO_DATE()` were not reporting that they could return `NULL`, so functions that invoked them might misinterpret their results. (Bug #14009)
- The example configuration files supplied with MySQL distributions listed the `thread_cache_size` variable as `thread_cache`. (Bug #13811)
- Using `ALTER TABLE` to add an index could fail if the operation ran out of temporary file space. Now it automatically makes a second attempt that uses a slower method but no temporary file. In this case, problems that occurred during the first attempt can be displayed with `SHOW WARNINGS`. (Bug #12166)
- The input polling loop for Instance Manager did not sleep properly. Instance Manager used up too much CPU as a result. (Bug #14388)

- Trying to take the logarithm of a negative value is now handled in the same fashion as division by zero. That is, it produces a warning when `ERROR_FOR_DIVISION_BY_ZERO` is set, and an error in strict mode. (Bug #13820)
- `LOAD DATA INFILE` would not accept the same character for both the `ESCAPED BY` and the `ENCLOSED BY` clauses. (Bug #11203)
- The value of `Last_query_cost` was not updated for queries served from the query cache. (Bug #10303)
- `TIMESTAMPDIFF()` returned an incorrect result if one argument but not the other was a leap year and a date was from March or later. (Bug #13534)
- The server incorrectly accepted column definitions of the form `DECIMAL(0, D)` for `D` less than 11. (Bug #13667)
- The displayed value for the `CHARACTER_MAXIMUM_LENGTH` column in the `INFORMATION_SCHEMA.COLUMNS` table was not adjusted for multi-byte character sets. (Bug #14290)
- A bugfix in MySQL 5.0.15 caused the displayed values for the `CHARACTER_MAXIMUM_LENGTH` and `CHARACTER_OCTET_LENGTH` columns in the `INFORMATION_SCHEMA.COLUMNS` table to be reversed. (Bug #14207)
- On Windows, the value of `character_sets_dir` in `SHOW VARIABLES` output was displayed inconsistently (using both `'/'` and `'\'` as pathname component separators). (Bug #14137)
- Subqueries in the `FROM` clause failed if the current database was `INFORMATION_SCHEMA`. (Bug #14089)
- Corrected a parser precedence problem that resulted in an `Unknown column ... in 'on clause'` error for some joins. (Bug #13832)
- For `LIKE ... ESCAPE`, an escape sequence longer than one character was accepted as valid. Now the sequence must be empty or one character long. If the `NO_BACKSLASH_ESCAPES SQL` mode is enabled, the sequence must be one character long. (Bug #12595)

- `SELECT DISTINCT CHAR(col_name)` returned incorrect results after `SET NAMES utf8`. (Bug #13233)
- A prepared statement failed with `Illegal mix of collations` if the client character set was `utf8` and the statement used a table that had a character set of `latin1`. (Bug #12371)
- Inserting a new row into an InnoDB table could cause `DATETIME` values already stored in the table to change. (Bug #13900)
- The default value of `query_prealloc_size` was set to 8192, lower than its minimum of 16384. The minimum has been lowered to 8192. (Bug #13334)
- The server did not take character set into account in checking the width of the `mysql.user.Password` column. As a result, it could incorrectly generate long password hashes even if the column was not long enough to hold them. (Bug #13064)
- Inserting `cp932` strings into a `VARCHAR` column caused a server crash rather than string truncation if the string was longer than the column definition. (Bug #12547)
- Two threads that were creating triggers on an InnoDB table at the same time could deadlock. (Bug #12739)
- **mysqladmin** and **mysqldump** would hang on SCO OpenServer. (Bug #13238)
- Where one stored procedure called another stored procedure: If the second stored procedure generated an exception, the exception was not caught by the calling stored procedure. For example, if stored procedure A used an `EXIT` statement to handle an exception, subsequent statements in A would be executed regardless when A was called by another stored procedure B, even if an exception that should have been handled by the `EXIT` was generated in A. (Bug #7049)
- Trying to create a stored routine with no database selected would crash the server. (Bug #13514, Bug #13587)
- Specifying `--default-character-set=cp-932` for **mysqld** would cause

SQL scripts containing comments written using that character set to fail with a syntax error. (Bug #13487)

- Trying to compile the server using the `--without-geometry` option caused the build to fail. (Bug #12991)

D.1.12. Changes in release 5.0.15 (19 October 2005: Production)

Functionality added or changed:

- **Warning: Incompatible change.** For `BINARY` columns, the pad value and how it is handled has changed. The pad value for inserts now is `0x00` rather than space, and there is no stripping of the pad value for selects. For details, see [Section 11.4.2, “The `BINARY` and `VARBINARY` Types”](#).
- **Warning: Incompatible change.** The `CHAR()` function now returns a binary string rather than a string in the connection character set. An optional `USING charset` clause may be used to produce a result in a specific character set instead. Also, arguments larger than 256 produce multiple characters. They are no longer interpreted modulo 256 to produce a single character each. These changes may cause some incompatibilities, as noted in [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#).
- `NDB Cluster`: The **`perror`** utility included with the `MySQL-Server` RPM now provides support for the `--ndb` option, and so can be used to obtain error message text for MySQL Cluster error codes. (Bug #13740)
- `NDB Cluster`: The **`ndb_mgm`** client now reports node startup phases automatically. (Bug #16197)
- When executing single-table `UPDATE` or `DELETE` queries containing an `ORDER BY ... LIMIT N` clause, but not having any `WHERE` clause, MySQL can now take advantage of an index to read the first N rows in the ordering specified in the query. If an index is used, only the first N records will be read, as opposed to scanning the entire table. (Bug #12915)
- The `MySQL-server` RPM now explicitly assigns the `mysql` system user to the `mysql` user group during the postinstallation process. This corrects an issue with upgrading the server on some Linux distributions whereby a

previously existing mysql user was not changed to the mysql group, resulting in wrong groups for files created following the installation. (Bug #12823)

- Added the `--tz-utc` option to **mysqldump**. This option adds `SET TIME_ZONE='+00:00'` to the dump file so that `TIMESTAMP` columns can be dumped and reloaded between servers in different time zones and protected from changes due to daylight saving time. (Bug #13052)
- When declaring a local variable (or parameter) named `password` or `name`, and setting it with `SET` (for example, `SET password = ''`), the new error message `ERROR 42000: Variable 'nnn' must be quoted with `...``, or `renamed` is returned (where `'nnn'` is `'password'` or `'names'`). This means there is a syntax conflict with special sentences like `SET PASSWORD = PASSWORD(...)` (for setting a user's password) and `set names default` (for setting charset and collation).

This must be resolved either by quoting the variable name: `SET `password` = ...`, which will set the local variable ``password``, or by renaming the variable to something else (if setting the user's password is the desired effect).

- The following statements now cause an implicit `COMMIT`:
 - `CREATE VIEW`
 - `ALTER VIEW`
 - `DROP VIEW`
 - `CREATE TRIGGER`
 - `DROP TRIGGER`
 - `CREATE USER`
 - `RENAME USER`
 - `DROP USER`

(Bug #13343)

- **NDBCluster**: A number of new or improved error messages have been implemented in this release in order to provide better and more accurate diagnostic information regarding cluster configuration issues and problems. (Bug #11739, Bug #11749, Bug #12044, Bug #12786, Bug #13197)
- **NDBCluster**: A new “smart” node allocation algorithm means that it is no longer necessary to use sequential IDs for cluster nodes, and that nodes not explicitly assigned IDs should now have IDs allocated automatically in most cases. In practical terms, this means that it is now possible to assign a set of node IDs such as 1, 2, 4, 5 without an error being generated due to the missing 3. (Bug #13009)

Bugs fixed:

- Issuing `STOP SLAVE` after having acquired a global read lock with `FLUSH TABLES WITH READ LOCK` caused a deadlock. Now `STOP SLAVE` generates an error in such circumstances. (Bug #10942)
- An expression in an `ORDER BY` clause failed with `Unknown column 'col_name' in 'order clause'` if the expression referred to a column alias. (Bug #11694)
- **mysqldump** could not dump views. (Bug #14061)
- Using an undefined variable in an `IF` or `SET` clause inside a stored routine produced an incorrect `unknown column ... in 'order clause'` error message. (Bug #13037)
- Trying to create a view dynamically using a prepared statement within a stored procedure failed with error 1295. (Bug #13095)
- **mysqldump --triggers** did not quote identifiers properly if the `--compatible` option was given, so the dump output could not be reloaded. (Bug #13146)
- Character set conversion was not being done for `FIND_IN_SET()`. (Bug #13751)

- `CAST(1E+300 TO SIGNED INT)` produced an incorrect result on little-endian machines. (Bug #13344)
- Corrected a memory-copying problem for `big5` values when using `icc` compiler on Linux IA-64 systems. (Bug #10836)
- On BSD systems, the system `crypt()` call could return an error for some salt values. The error was not handled, resulting in a server crash. (Bug #13619)
- Character set file parsing during `mysql_real_connect()` read past the end of a memory buffer. (Bug #6413)
- InnoDB: Queries that were executed using an `index_merge` union or intersection could produce incorrect results if the underlying table used the InnoDB storage engine and had a primary key containing `VARCHAR` members. (Bug #13484)
- `CREATE DEFINER=... VIEW ...` caused the server to crash when run with `--skip-grant-tables`. (Bug #13504)
- The `--interactive-timeout` and `--slave-net-timeout` options for **mysqld** were not being obeyed on Mac OS X and other BSD-based platforms. (Bug #8731)
- Queries of the form `(SELECT ...) ORDER BY ...` were being treated as a `UNION`. This improperly resulted in only distinct values being returned (because `UNION` by default eliminates duplicate results). Also, references to column aliases in `ORDER BY` clauses following parenthesized `SELECT` statements were not resolved properly. (Bug #7672)
- If special characters such as `'_'`, `'%'`, or the escape character were included within the prefix of a column index, `LIKE` pattern matching on the indexed column did not return the correct result. (Bug #13046, Bug #13919)
- An `UPDATE` query using a join would be executed incorrectly on a replication slave. (Bug #12618)
- Server crashed during a `SELECT` statement, writing a message like this to the

error log:

```
InnoDB: Error: MySQL is trying to perform a SELECT  
InnoDB: but it has not locked any tables in ::external_lock()!
```

(Bug #12736)

- NDBCluster: **ndb_mgmd** would allow a node to be stopped or restarted while another node was still starting up, which could crash the cluster. It should now not be possible to issue a node stop or restart while a different node is still restarting, and the cluster management client issues an error if an attempt is made to do so. (Bug #13461)
- NDBCluster: Placing multiple [TCP DEFAULT] sections in the cluster config.ini file crashed **ndb_mgmd**. (The **ndb_mgmd** process now exits gracefully with an appropriate error message instead.) (Bug #13611)
- NDBCluster: Trying to run **ndbd** as system root when connecting to a **mysqld** process running as the mysql system user via SHM caused the **ndbd** process to crash. (**ndbd** should now exit gracefully with an appropriate error message instead.) (Bug #9249)
- Server may over-allocate memory when performing a FULLTEXT search for stopwords only. (Bug #13582)
- Queries that use indexes in normal SELECT statements may cause range scans in VIEWS. (Bug #13327)
- When calling a stored procedure with the syntax CALL schema.procedurename and no default schema selected, ERROR 1046 was displayed after the procedure returned. (Bug #13616)
- With --log-slave-updates Exec_master_log_pos of SQL thread lagged IO (Bug #13023)
- SHOW CREATE TABLE did not display any FOREIGN KEY clauses if a temporary file could not be created. Now SHOW CREATE TABLE displays an error message in an SQL comment if this occurs. (Bug #13002)
- A column in the ON condition of a join that referenced a table in a nested join could not be resolved if the nested join was a right join. (Bug #13597)

- A qualified reference to a view column in the `HAVING` clause could not be resolved. (Bug #13410)
- **comp_err** did not detect when multiple error messages for a language were given for an error symbol. (Bug #13071)
- For XA transaction IDs (*gtrid.bqual.formatID*), uniqueness is supposed to be assessed based on *gtrid* and *bqual*. MySQL was also including *formatID* in the uniqueness check. (Bug #13143)
- Local (non-XA) and XA transactions are supposed to be mutually exclusive within a given client connection, but this prohibition was not always enforced. (Bug #12935)
- **mysqlcheck --all-databases --analyze --optimize** failed because it also tried to analyze and optimize the `INFORMATION_SCHEMA` tables which it can't. (Bug #13783)
- `SELECT * INTO OUTFILE ... FROM INFORMATION_SCHEMA.schemata` failed with an Access denied error. (Bug #13202)
- A table or view named Ç (C-cedilla) couldn't be dropped. (Bug #13145)
- Tests containing `SHOW TABLE STATUS` or `INFORMATION_SCHEMA` failed on `opnsrv6c`. (Bug, #14064, Bug #14065)

D.1.13. Changes in release 5.0.14 (Not released)

Functionality added or changed:

The limit of 255 characters on the input buffer for **mysql** on Windows has been lifted. The exact limit depends on what the system allows, but can be up to 64K characters. A typical limit is 16K characters. (Bug #12929)

Re-enabled the `--delayed-inserts` option for **mysqldump**, which now checks for each table dumped whether its storage engine supports `DELAYED` inserts. (Bug #7815)

Added the `myisam_stats_method`, which controls whether `NULL` values in

indexes are considered the same or different when collecting statistics for MyISAM tables. This influences the query optimizer as described in [Section 7.4.7, “MyISAM Index Statistics Collection”](#). (Bug #12232)

- When an InnoDB foreign key constraint is violated, the error message now indicates which table, column, and constraint names are involved. (Bug #3443)
- Configure-time checking for the availability of multi-byte macros and functions in the bundled `readline` library. This improves handling of multi-byte character sets in the `mysql` client. (Bug #3982)
- The `CHAR()` function now takes into account the character set and collation given by the `character_set_connection` and `collation_connection` system variables. For an argument n to `CHAR()`, the result is $n \bmod 256$ for single-byte character sets. For multi-byte character sets, n must be a valid code point in the character set. Also, the result string from `CHAR()` is checked for well-formedness. For invalid arguments, or a result that is not well-formed, MySQL generates a warning (or, in strict SQL mode, an error). (Bug #10504)
- `RENAME TABLE` now works for views as well, as long as you do not try to rename a view into a different database. (Bug #5508)
- Multiple-table `UPDATE` and `DELETE` statements that do not affect any rows are now written to the binary log and will replicate. (Bug #13348, Bug #12844)
- Range scans can now be performed for queries on VIEWS such as `column IN (<constants>)` and `column BETWEEN ConstantA AND ConstantB`. (Bug #13317)

Bugs fixed:

- `NDBCluster`: A trigger updating the value of an `AUTO_INCREMENT` column in a Cluster table would insert an error code rather than the expected value into the column. (Bug #13961)
- `NDBCluster`: When performing a delete of a great many (tens of thousands of) rows at once from a Cluster table, an improperly dereferenced pointer

could cause the **mysqld** process to crash. (Bug #9282)

- CHECKSUM TABLE locked InnoDB tables and did not use a consistent read. (Bug #12669)
- The --skip-innodb-doublewrite option disables use of the InnoDB doublewrite buffer. However, having this option in effect when creating a new MySQL installation prevented the buffer from even being created, resulting in a server crash later. (Bug #13367)
- MySQL programs in binary distributions for Solaris 8/9/10 x86 systems would not run on Pentium III machines. (Bug #6772)
- When SELECT ... FOR UPDATE or SELECT ... LOCK IN SHARE MODE for an InnoDB table were executed from within a stored function or a trigger, they were converted to a non-locking consistent read. (Bug #11238)
- NDB Cluster: If **ndb_restore** could not find a free **mysqld** process, it crashed. (Bug #13512)
- NDB Cluster: Receipt of several enter single user mode commands by multiple **ndb_mgmd** processes within a short period of time resulted in cluster shutdown. (Bug #13053)
- NDB Cluster: Multiple **ndb_mgmd** processes in a cluster would not know each other's IP addresses. (Bug #12037)
- NDB Cluster: With two **mgmd** processes in a cluster, **ndb_mgmd** output for SHOW would display the same IP address for both processes, even when they were on different hosts. (Bug #11595)
- NDB Cluster: Queries on NDB tables that are executed using index_merge/union or index_merge/intersection could produce incorrect results. (Bug #13081)
- The --replicate-rewrite-db and --replicate-do-table options did not work for statements in which tables were aliased to names other than those listed by the options. (Bug #11139)
- After running **configure** with the --with-embedded-privilege-control

option, the embedded server failed to build. (Bug #13501)

- Nested handlers within stored procedures didn't work. (Bug #6127)
- The optimizer chose a less efficient execution plan for `col_name BETWEEN const AND const` than for `col_name = const`, even though the two expressions are logically equivalent. Now the optimizer can use the ref access method for both expressions. (Bug #13455)
- Incorrect creation of DECIMAL local variables in a stored procedure could cause a server crash. (Bug #12589)
- Queries against a MERGE table that has a composite index could produce incorrect results. (Bug #9112)
- The server was not rejecting `FLOAT(M,D)` or `DOUBLE(M,D)` columns specifications when *M* was less than *D*. (Bug #12694)
- After running **configure** with the `--without-server` option, the distribution failed to build. (Bug #11680, Bug #13550)
- Joins nested under NATURAL or USING joins were sometimes not initialized properly, causing a server crash. (Bug #13545)
- Locking a view with the query cache enabled and `query_cache_wlock_invalidate` enabled could cause a server crash. (Bug #13424)
- A HAVING clause that references an unqualified view column name could crash the server. (Bug #13411)
- Comparisons involving row constructors containing constants could cause a server crash. (Bug #13356)
- NDB Cluster: LOAD DATA INFILE with a large data file failed. (Bug #10694)
- NDB Cluster: Adding an index to a table with a large number of columns (more than 100) crashed the storage node. (Bug #13316)

- Calling the `FORMAT()` function with a `DECIMAL` column value caused a server crash when the value was `NULL`. (Bug #13361)
- Aggregate functions sometimes incorrectly were allowed in the `WHERE` clause of `UPDATE` and `DELETE` statements. (Bug #13180)
- It was possible to create a view that executed a stored function for which you did not have the `EXECUTE` privilege. (Bug #12812)
- `BIT` columns and following columns in `NDB` tables were corrupt when dumped by **mysqldump**. (Bug #13152)
- `NATURAL` joins and joins with `USING` against a view could return `NULL` rather than the correct value. (Bug #13127)
- Use of a user-defined function within the `HAVING` clause of a query resulted in an `Unknown column` error. (Bug #11553)
- For queries for which the optimizer determined a join type of “Range checked for each record” (as shown by `EXPLAIN`, the query sometimes could cause a server crash, depending on the data distribution. (Bug #12291)
- For queries with `DISTINCT` and `WITH ROLLUP`, the `DISTINCT` should be applied after the rollup operation, but was not always. (Bug #12887)
- The server crashed when processing a view that invoked the `CONVERT_TZ()` function. (Bug #11416)
- Shared-memory connections were not working on Windows. (Bug #12723)

D.1.14. Changes in release 5.0.13 (22 September 2005: Release Candidate)

Functionality added or changed:

- The syntax for `CREATE VIEW` and `ALTER VIEW` statements now includes `DEFINER` and `SQL SECURITY` clauses for specifying the security context to be used when checking access privileges at view invocation time. (The syntax is present in 5.0.13, but these clauses have no effect until 5.0.16.) See [Section 19.2, “CREATE VIEW Syntax”](#), for more information.

- The `--hex-dump` option for **mysqldump** now also applies to BIT columns.
- Added a `--routines` option for **mysqldump** that enables dumping of stored routines. (Bug #9056)
- The connection string for FEDERATED tables now is specified using a `CONNECTION` table option rather than a `COMMENT` table option.
- Better detection of connection timeout for replication servers on Windows allows elimination of extraneous `Lost connection` errors in the error log. (Bug #5588)
- The counters for the `Key_read_requests`, `Key_reads`, `Key_write_requests`, and `Key_writes` status variables were changed from unsigned long to unsigned longlong to accommodate larger values before the variables roll over and restart from 0. (Bug #12920)
- The restriction on the use of `PREPARE`, `EXECUTE`, and `DEALLOCATE PREPARE` within stored procedures was lifted. The restriction still applies to stored functions and triggers. (Bug #10975, Bug #7115, Bug #10605)
- A new command line argument was added to **mysqld** to ignore client character set information sent during handshake, and use server side settings instead, to reproduce 4.0 behavior (Bug #9948):

`mysqld --skip-character-set-client-handshake`
- `OPTIMIZE TABLE` and `HANDLER` now are prohibited in stored procedures and functions and in triggers. (Bug #12953, Bug #12995)
- InnoDB: The `TRUNCATE TABLE` statement for InnoDB tables always resets the counter for an `AUTO_INCREMENT` column now, regardless of whether there is a foreign key constraint on the table. (Beginning with 5.0.3, `TRUNCATE TABLE` reset the counter, but only if there was no such constraint.) (Bug #11946)
- The `LEAST()` and `GREATEST()` functions used to return NULL only if all arguments were NULL. Now they return NULL if any argument is NULL, the same as Oracle. (Bug #12791)

- Two new collations have been added for Esperanto: `utf8_esperanto_ci` and `ucs2_esperanto_ci`.
- Reorder network startup to come after all other initialization, particularly storage engine startup which can take a long time. This also prevents MySQL from being run on a privileged port (any port under 1024) unless run as the root user. (Bug #11707)
- The Windows binary packages are now compiled with the Microsoft Visual Studio 2003 compiler instead of Microsoft Visual C++ 6.0.
- The binaries compiled with the Intel icc compiler are now built using icc 9.0 instead of icc 8.1. You will have to install new versions of the Intel icc runtime libraries, which are available from here: (<http://dev.mysql.com/downloads/os-linux.html>)

Bugs fixed:

- **Incompatible change:** A lock wait timeout caused InnoDB to roll back the entire current transaction. Now it rolls back only the most recent SQL statement. (Bug #12308)
- The FEDERATED storage engine does not support ALTER TABLE, but no appropriate error message was issued. (Bug #13108)
- **mysqldump** did not dump triggers properly. (Bug #12597)
- **NDBCluster:** The average row size for Cluster tables was being calculated incorrectly. This affected the values shown for the `Data_length` and `Avg_row_length` columns in the output generated by `SHOW TABLE STATUS` as well as the values for the `data_length` and `data_length/table_rows` columns shown in the TABLES table of the INFORMATION_SCHEMA database with respect to Cluster tables (tables using other storage engines were not affected by this bug). (Bug #9896)
- Within a stored procedure, fetching a large number of rows in a loop using a cursor could result in a server crash or an out of memory error. Also, values inserted within a stored procedure using a cursor were interpreted as `latin1` even if character set variables had been set to a different character set. (Bug #6513, Bug #9819)

- For a server compiled with yaSSL, clients that used MySQL Connector/J were not able to establish SSH connections. (Bug #13029)
- When used in view definitions, DAYNAME(expr), DAYOFWEEK(expr), WEEKDAY(expr) were incorrectly treated as though the expression was TO_DAYS(expr) or TO_DAYS(TO_DAYS(expr)). (Bug #13000)
- Incorrect implicit nesting of joins caused the parser to fail on queries of the form SELECT ... FROM t1 JOIN t2 JOIN t3 ON t1.t1col = t3.t3col with an Unknown column 't1.t1col' in 'on clause' error. (Bug #12943)
- NDB: A cluster shutdown following the crash of a data node would fail to terminate the remaining node processes, even though **ndb_mgm** showed the shutdown request as having been completed. (Bug #10938, Bug #9996, Bug #11623)
- A column that can be NULL was not handled properly for WITH ROLLUP in a subquery or view. (Bug #12885)
- Within a transaction, the following statements now cause an implicit commit: CREATE FUNCTION, DROP FUNCTION, DROP PROCEDURE, ALTER FUNCTION, ALTER PROCEDURE, CREATE PROCEDURE. This corrects a problem where these statements followed by ROLLBACK might not be replicated properly. (Bug #12870)
- Simultaneous execution of DML statements and CREATE TRIGGER or DROP TRIGGER statements on the same table could cause server crashes or errors. (Bug #12704)
- If a stored function invoked from a SELECT failed with an error, it could cause the client connection to be dropped. Now such errors generate warnings instead so as not to interrupt the SELECT. (Bug #12379)
- A concurrency problem for CREATE ... SELECT could cause a server crash. (Bug #12845)
- The server incorrectly generated an Unknown table error message when for attempts to drop tables in the INFORMATION_SCHEMA database. Now it issues an Access denied message. (Bug #9846)

- The server allowed privileges to be granted explicitly for the INFORMATION_SCHEMA database. Such privileges are always implicit and should not be grantable. (Bug #10734)
- The server allowed TEMPORARY tables and stored procedures to be created in the INFORMATION_SCHEMA database. (Bug #9683, Bug #10708)
- The server failed to disallow SET AUTOCOMMIT in stored functions and triggers. It is allowed to change the value of AUTOCOMMIT in stored procedures, but a runtime error might occur if the procedure is invoked from a stored function or trigger. (Bug #12712)
- Using an INOUT parameter with a DECIMAL data type in a stored procedure caused a server crash. (Bug #12979)
- Performing an IS NULL check on the MIN() or MAX() of an indexed column in a complex query could produce incorrect results. (Bug #12695)
- The mysql.server script contained incorrect path for the libexec directory. (Bug #12550)
- The NDB START BACKUP command could be interrupted by a SHOW command. (Bug #13054)
- The LIKE ... ESCAPE syntax produced invalid results when escape character was larger than one byte. (Bug #12611)
- A client connection thread cleanup problem caused the server to crash when closing the connection if the binary log was enabled. (Bug #12517)
- Using AS to rename a column selected from a view in a subquery made it not possible to refer to that column in the outer query. (Bug #12993)
- The character_set_system system variable could not be selected with SELECT @@character_set_system. (Bug #11775)
- A view-creation statement of the form CREATE VIEW name AS SELECT ... FROM tbl_name AS name failed with a Not unique table/alias: 'name' error. (Bug #6808)

- UNION [DISTINCT] was not removing all duplicates for multi-byte character values. (Bug #12891)
- Multiplying a DECIMAL value within a loop in a stored routine could incorrectly result in a value of NULL. (Bug #12938)
- **mysql** and **mysqldump** were ignoring the `--defaults-extra-file` option. (Bug #12917)
- Columns named in the USING() clause of JOIN ... USING() were incorrectly resolved in case-sensitive fashion. (Bug #13067)
- Local variables in stored routines were not always initialized correctly. (Bug #13133)
- SHOW FIELDS FROM *schemaname.viewname* caused error 1046 when no default schema was set. (Bug #12905)
- The value of `character_set_results` could be set to NULL, but returned the string "NULL" when retrieved. (Bug #12363)
- InnoDB: Limit recursion depth to 200 in deadlock detection to avoid running out of stack space. (Bug #12588)
- GROUP_CONCAT() ignored an empty string if it was the first value to occur in the result. (Bug #12863)
- Outer join elimination was erroneously applied for some queries that used a NOT BETWEEN condition, an IN(value_list) condition, or an IF() condition. (Bug #12101, Bug #12102)
- SHOW FIELDS truncated the TYPE column to 40 characters. (Bug #7142)
- Use of PREPARE and EXECUTE with a statement that selected from a view in a subquery could cause a server crash. (Bug #12651)
- On HP-UX 11.x (PA-RISC), the -L option caused **mysqlimport** to crash. (Bug #12958)
- If the binary log is enabled, execution of a stored procedure that modifies

table data and uses user variables could cause a server crash or incorrect information to be written to the binary log. (Bug #12637)

- Queries with subqueries, where the inner subquery uses the range or `index_merge` access method, could return incorrect results. (Bug #12720)
- After changing the character set with `SET CHARACTER SET`, the result of the `GROUP_CONCAT()` function was not converted to the proper character set. (Bug #12829)
- A bug introduced in MySQL 5.0.12 caused `SHOW TABLE STATUS` to display an `Auto_increment` value of 0 for InnoDB tables. (Bug #12973)
- Foreign keys were not properly enforced in `TEMPORARY` tables. Foreign keys now are disallowed in `TEMPORARY` tables. (Bug #12084)
- Replication of `LOAD DATA INFILE` failed between systems that use different pathname syntax (such as delimiter characters). (Bug #11815)
- Within a stored procedure, a server crash was caused by assigning to a `VARCHAR INOUT` parameter the value of an expression that included the variable itself. (For example, `SET c = c.`) (Bug #12849)
- `SELECT ... JOIN ... ON ... JOIN ... USING` caused a server crash. (Bug #12977)
- Using `GROUP BY` when selecting from a view in some cases could cause incorrect results to be returned. (Bug #12922)
- **myisampack** did not properly pack BLOB values larger than 2^{24} bytes. (Bug #4214)
- Incorrect results could be returned from a view processed using a temporary table. (Bug #12941)
- The server crashed when one thread resized the query cache while another thread was using it. (Bug #12848)
- **mysqld_multi** now quotes arguments on command lines that it constructs to avoid problems with arguments that contain shell metacharacters. (Bug

#11280)

- InnoDB: A consistent read could return inconsistent results due to a bug introduced in MySQL 5.0.5. (Bug #12947)
- Deadlock occurred when several account management statements were run (particularly between FLUSH PRIVILEGES/SET PASSWORD and GRANT/REVOKE statements). (Bug #12423)
- The Windows installer made a change to one of the `mysql.proc` table files, causing stored routine functionality to be compromised. The Windows installer now never overwrites files in the MySQL data directory. During an upgrade from one version to another, a file in the data directory will not be overwritten even if it has not been modified since it was put there by an older installer.

If you have already lost access to stored routines because of this problem, you can get them back using the following procedure:

- Stop the server.
- In the `mysql\data` directory under your MySQL installation directory, and replace the `proc.frm` file with corresponding file from the version of MySQL that you were using before you upgraded.
- Start the server
- Start the **mysql** command-line client (use the `root` account or another account that has full database privileges) and execute the `mysql_fix_privilege_tables.sql` script that upgrades the grant tables to the current structure. Instructions for doing this are given in [Section 5.6.1, “mysql fix privilege tables — Upgrade MySQL System Tables”](#).

After this, all stored routine functionality should work. (Bug #12820)

- On Windows, the server was preventing tables from being created if the table name was a prefix of a forbidden name. For example, `nu1` is a forbidden name because it's the same as a Windows device name, but a table with the name of `n` or `nu` was being forbidden as well. (Bug #12325)

- InnoDB was too permissive with `LOCK TABLE ... READ LOCAL` and allowed new inserts into the table. Now `READ LOCAL` is equivalent to `READ` for InnoDB. This will cause slightly more locking in **mysqldump**, but makes InnoDB table dumps consistent with MyISAM table dumps. (Bug #12410)
- Use of the **mysql** client `HELP` command from within a stored routine caused a “packets out of order” error and a lost connection. Now `HELP` is detected and disallowed within stored routines. (Bug #12490)
- Use of yaSSL for a secure client connection caused `LOAD DATA LOCAL INFILE` to fail. (Bug #11286)
- `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` no longer qualify the routine name with the database name, for consistency with the behavior of `SHOW CREATE TABLE`. (Bug #10362)
- A `UNION` of long `utf8 VARCHAR` columns was sometimes returned as a column with a `LONGTEXT` data type rather than `VARCHAR`. This could prevent such queries from working at all if selected into a `MEMORY` table because the `MEMORY` storage engine does not support the `TEXT` data types. (Bug #12537)
- If a client has opened an InnoDB table for which the `.ibd` file is missing, InnoDB would not honor a `DROP TABLE` statement for the table. (Bug #12852)
- `ALTER TABLE ... DISCARD TABLESPACE` for non-InnoDB table caused the client to lose the connection. (The server was not returning the error properly.) (Bug #12207)
- `DO IFNULL(NULL, NULL)` and `SELECT CAST(IFNULL(NULL, NULL) AS DECIMAL)` caused a server crash. (Bug #12841)
- When using a cursor, a `SELECT` statement that uses a `GROUP BY` clause could return incorrect results. (Bug #11904)
- The `SYSDATE()` function now returns the time at which it was invoked. In particular, within a stored routine or trigger, `SYSDATE()` returns the time at which it executes, not the time at which the stored routine or triggering statement began to execute. (Bug #12480)

- `CREATE VIEW` inside a stored procedure caused a server crash if the table underlying the view had been deleted. (Bug #12468)
- A memory leak resulting from repeated `SELECT ... INTO` statements inside a stored procedure could cause the server to crash. (Bug #11333)

D.1.15. Changes in release 5.0.12 (02 September 2005)

Functionality added or changed:

- **Incompatible change:** Beginning with MySQL 5.0.12, natural joins and joins with `USING`, including outer join variants, are processed according to the SQL:2003 standard. The changes include elimination of redundant output columns for `NATURAL` joins and joins specified with a `USING` clause and proper ordering of output columns. (Bug #6136, Bug #6276, Bug #6489, Bug #6495, Bug #6558, Bug #9067, Bug #9978, Bug #10428, Bug #10646, Bug #10972.) The precedence of the comma operator also now is lower compared to `JOIN`. (Bug #4789, Bug #12065, Bug #13551.)

These changes make MySQL more compliant with standard SQL. However, they can result in different output columns for some joins. Also, some queries that appeared to work correctly prior to 5.0.12 must be rewritten to comply with the standard. For details about the scope of the changes and examples that show what query rewrites are necessary, see [Section 13.2.7.1, “JOIN Syntax”](#).

- Recursive triggers are detected and disallowed. Also, within a stored function or trigger, it is not allowable to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger. (Bug #11896, Bug #12644)
- `SHOW TABLE STATUS` for a view now shows `VIEW` in uppercase, consistent with `SHOW TABLES` and `INFORMATION_SCHEMA`. (Bug #5501)
- An optimizer estimate of zero rows for a non-empty InnoDB table used in a left or right join could cause incomplete rollback for the table. (Bug #12779)
- Calls to stored procedures were written to the binary log even within

transactions that were rolled back, causing them to be executed on replication slaves. (Bug #12334)

- Interleaved execution of stored procedures and functions could be written to the binary log incorrectly, causing replication slaves to get out of sync. (Bug #12335)
- A query of the form `SHOW TABLE STATUS FROM db_name WHERE name IN (select_query)` would crash the server. (Bug #12636)
- Users created using an IP address or other alias rather than a hostname listed in `/etc/hosts` could not set their own passwords. (Bug #12302)
- Using `DESCRIBE` on a view after renaming a column in one of the view's base tables caused the server to crash. (Bug #12533)
- `SHOW OPEN TABLES` now supports `FROM` and `LIKE` clauses. (Bug #12183)
- `SHOW TABLE STATUS FROM INFORMATION_SCHEMA` now sorts output by table name the same as it does for other databases. (Bug #12315)
- `SHOW ENGINE INNODB STATUS` now can display longer query strings. (Bug #7819)
- Added the `SLEEP()` function, which pauses for the number of seconds given by its argument. (Bug #6760)
- Trying to drop the default keycache by setting `@@global.key_buffer_size` to zero now returns a warning that the default keycache cannot be dropped. (Bug #10473)
- The stability of cursors when used with InnoDB tables was greatly improved. (Bug #11832, Bug #12243, Bug #11309)
- It is no longer possible to issue `FLUSH` commands from within stored functions or triggers. See [Section I.1, “Restrictions on Stored Routines and Triggers”](#), for details. (Bug #12280, Bug #12307)
- `INFORMATION_SCHEMA` objects are now reported as a `SYSTEM VIEW` table type. (Bug #11711)

Bugs fixed:

- CHECKSUM TABLE command returned incorrect results for tables with deleted rows. After upgrading, users who used stored checksum information to detect table changes should rebuild their checksum data. (Bug #12296)
- A data type of CHAR BINARY was not recognized as valid for stored routine parameters. (Bug #9048)
- SET GLOBAL TRANSACTION ISOLATION LEVEL was not working. (Bug #11207)
- NDB Cluster: Corrected the parsing of the CLUSTERLOG command by **ndb_mgm** to allow multiple items. (Bug #12833)
- NDB Cluster: Improved error messages related to filesystem issues. (Bug #11218)
- NDB Cluster: When a schema was detected to be corrupt, **ndb** neglected to close it, resulting in a “file already open” error if the schema was opened again later. (Bug #12027)
- NDB Cluster: When it could not copy a fragment, **ndbd** exited without printing a message about the condition to the error log. Now the message is written. (Bug #12900)
- NDB Cluster: When a disk full condition occurred, **ndbd** exited without printing a message about the condition to the error log. Now the message is written. (Bug #12716)
- mysql_fix_privilege_tables.sql was missing a comma, causing a syntax error when executed. (Bug #12705)
- STRCMP() was not handled correctly in views. (Bug #12489)
- NDB Cluster: Bad values in config.ini caused **ndb_mdmd** to crash. (Bug #12043)
- TRUNCATE TABLE did not work with TEMPORARY InnoDB tables. (Bug #11816)

- Built-in commands for the **mysql** client, such as `delimiter` and `\d` are now always parsed within files that are read using the `\.` and `source` commands. (Bug #11523)
- `ALTER TABLE db_name.t RENAME t` did not move the table to default database unless the new name was qualified with the database name. (Bug #11493)
- It was not possible to create a stored function with a spatial return value data type. (Bug #10499)
- The only valid values for the `PACK_KEYS` table option are 0 and 1, but other values were being accepted. (Bug #10056)
- If a `DROP DATABASE` fails on a master server due to the presence of a non-database file in the database directory, the master have the database tables deleted, but not the slaves. To deal with failed database drops, we now write `DROP TABLE` statements to the binary log for the tables so that they are dropped on slaves. (Bug #4680)
- Improper use of loose index scan in InnoDB sometimes caused incorrect query results. (Bug #12672)
- `DELETE` or `UPDATE` for an indexed MyISAM table could fail. This was due to a change in end-space comparison behavior from 4.0 to 4.1. (Bug #12565)
- Joins on `VARCHAR` columns of different lengths could produce incorrect results. (Bug #11398)
- A “Duplicate column name” error no longer occurs when selecting from a view defined as `SELECT *` from a join that uses a `USING` clause on tables that have a common column name. (Bug #6558)
- Invocations of the `SLEEP()` function incorrectly could get optimized away for statements in which it occurs. Statements containing `SLEEP()` incorrectly could be stored in the query cache. (Bug #12689)
- NDB Cluster: An `ALTER TABLE` command caused loss of data stored prior to the issuing of the command. (Bug #12118)

- Query cache is switched off if a thread (connection) has tables locked. This prevents invalid results where the locking thread inserts values between a second thread connecting and selecting from the table. (Bug #12385)
- `NOW()`, `CURRENT_TIME` and values generated by timestamp columns are now constant for the duration of a stored function or trigger. This prevents the breaking of statements-based replication. (Bug #12480, Bug #12481)
- Some statements executed on a master server caused the SQL thread on a slave to run out of memory. (Bug #12532)
- A `SELECT DISTINCT` query with a constant value for one of the columns would return only a single row. (Bug #12625)
- NDB cluster: Cluster failed to take character set data into account when recomputing hashes (and thus could not locate records for updating or deletion) following a configuration change and node restart. (Bug #12220)
- NDB cluster: Wrong error message displayed when cluster management server closed port while **mysqld** was connecting. (Bug #10950)
- A view was allowed to depend on a function that referred to a temporary table. (Bug #10970)
- Prepared statement parameters could cause errors in the binary log if the character set was cp932. (Bug #11338)
- The `CREATE_OPTIONS` column of `INFORMATION_SCHEMA.TABLES` showed incorrect options for tables in `INFORMATION_SCHEMA`. (Bug #12397)
- `MEMORY` tables using B-Tree index on 64-bit platforms could produce false table is full errors. (Bug #12460)
- Issuing `FLUSH INSTANCES` followed by `STOP INSTANCE` caused instance manager to crash. (Bug #10957)
- Duplicate instructions in stored procedures resulted in incorrect execution when the optimizer optimized the duplicate code away. (Bug #12168)
- `SHOW TABLES FROM` returned wrong error message if the schema specified

did not exist. (Bug #12591)

- The ROW() function returned an incorrect result when comparison involved NULL values. (Bug #12509)
- Views with multiple UNION and UNION ALL produced incorrect results. (Bug #10624)
- Stored procedures with particularly long loops could crash server due to memory leak. (Bug #12297, Bug #11247)
- Trigger and stored procedure execution could break replication. (Bug #12482)
- A server crash could result from an update of a view defined as a join, even though the update updated only a single table. (Bug #12569)
- On Windows when the --innodb_buffer_pool_awesome_mem_mb option has been given, the server detects whether AWE support is available and has been compiled into the server, and displays an appropriate error message if not. (Bug #6581)
- The NUMERIC_SCALE column of the INFORMATION_SCHEMA.COLUMNS table should be returned as 0 for integer columns. It was being returned as NULL. (Bug #12301)
- The COLUMN_DEFAULT column of the INFORMATION_SCHEMA.COLUMNS table should be returned as NULL if a column has no default value. An empty string was being returned if the column was defined as NOT NULL. (Bug #12518)
- Slave I/O threads were considered to be in the running state when launched (rather than after successfully connecting to the master server), resulting in incorrect SHOW SLAVE STATUS output. (Bug #10780)
- Column names in subqueries must be unique, but were not being checked for uniqueness. (Bug #11864)
- On Windows, the server could crash during shutdown if both replication threads and normal client connection threads were active. (Bug #11796)

- Some subqueries of the form `SELECT ... WHERE ROW(...) IN (subquery)` were being handled incorrectly. (Bug #11867)
- Selecting from a view after `INSERT` statements for the view's underlying table yielded different results than subsequent selects. (Bug #12382)
- The `mysql_info()` C API function could return incorrect data when executed as part of a multi-statement that included a mix of statements that do and do not return information. (Bug #11688)
- When restoring `INFORMATION_SCHEMA` as the default database after failing to execute a stored procedure in an inaccessible database, the server returned a spurious `ERROR 42000: Unknown database 'information_schema'` message. (Bug #12318)
- Renamed the `rest()` macro in `my_list.h` to `list_rest()` to avoid name clashes with user code. (Bug #12327)
- `DATE_ADD()` and `DATE_SUB()` were converting invalid dates to `NULL` in `TRADITIONAL SQL` mode rather than rejecting them with an error. (Bug #10627)
- A trigger that included a `SELECT` statement could cause a server crash. (Bug #11587)
- An incorrect conversion from `double` to `ulonglong` caused indexes not to be used for `BDB` tables on `HP-UX`. (Bug #10802)
- **myisampack** failed to delete `.TMD` temporary files when run with `-T` option. (Bug #12235)
- Added portability check for Intel compiler to address a problem compiling `InnoDB` code. (Bug #11510)
- `XA` allowed two active transactions to be started with the same `XID`. (Bug #12162)
- Concatenating `USER()` or `DATEBASE()` with a column produced invalid results. (Bug #12351)

- Creating a view that included the `TIMESTAMPDIFF()` function resulted in an invalid view. (Bug #12298)
- Comparison of InnoDB multi-part primary keys that include `VARCHAR` columns can result in incorrect results. (Bug #12340)
- For PKG installs on Mac OS X, the preinstallation and postinstallation scripts were being run only for new installations and not for upgrade installations, resulting in an incomplete installation process. (Bug #11380)
- Using cursors and nested queries for the same table, corrupted results were returned for the outer query. (Bug #11909)
- User variables were not automatically cast for comparisons, causing queries to fail if the column and connection character sets differed. Now when mixing strings with different character sets but the same coercibility, allow conversion if one character set is a superset of the other. (Bug #10892)
- Selecting from a view defined as a join over many tables could result in a server crash due to miscalculation of the number of conditions in the `WHERE` clause. (Bug #12470)
- Pathname values for options such as `---basedir` or `--datadir` didn't work on Japanese Windows machines for directory names containing multi-byte characters having a second byte of `0x5C` (`'\'`). (Bug #5439)
- A race condition between server threads could cause a crash if one thread deleted a stored routine while another thread was executing a stored routine. (Bug #12228)
- Mishandling of comparison for rows containing `NULL` values against rows produced by an `IN` subquery could cause a server crash. (Bug #12392)
- Inserting `NULL` into a `GEOMETRY` column for a table that has a trigger could result in a server crash if the table was subsequently dropped. (Bug #12281)
- A failure to obtain a lock for an `IN SHARE MODE` query could result in a server crash. (Bug #12082)
- `SELECT ... INTO var_name` within a trigger could cause a server crash.

(Bug #11973)

- INSERT ... SELECT ... ON DUPLICATE KEY UPDATE could fail with an erroneous “Column '*col_name*' specified twice” error. (Bug #10109)
- SHOW TABLE STATUS sometimes reported a Row_format value of Dynamic for MEMORY tables, though such tables always have a format of Fixed. (Bug #3094)
- A query using a LEFT JOIN, an IN subquery on the outer table, and an ORDER BY clause, caused the server to crash when cursors were enabled. (Bug #11901)
- Using a stored procedure that referenced tables in the INFORMATION_SCHEMA database would return an empty result set. (Bug #10055, Bug #12278)
- Columns defined as TINYINT(1) were redefined as TINYINT(4) when incorporated into a VIEW. (Bug #11335)
- ISO-8601 formatted dates were not being parsed correctly. (Bug #7308)
- FLUSH TABLES WITH READ LOCK combined with LOCK TABLE .. WRITE caused deadlock. (Bug #9459)
- NULL column definitions read incorrectly for inner tables of nested outer joins. (Bug #12154)
- GROUP_CONCAT ignores the DISTINCT modifier when used in a query joining multiple tables where one of the tables has a single row. (Bug #12095)
- UNION query with FULLTEXT could cause server crash. (Bug #11869)

D.1.16. Changes in release 5.0.11 (06 August 2005)

Functionality added or changed:

- Security improvement: Applied a patch that addresses a potential zlib data vulnerability that could result in an application crash. ([CVE-2005-1849](#)) This only affects the binaries for platforms that are linked statically against the bundled zlib (most notably Microsoft Windows and HP-UX).

- `SHOW CHARACTER SET` and `INFORMATION_SCHEMA` now properly report the Latin1 character set as cp1252. (Bug #11216)
- **mysqldump** now dumps triggers for each dumped table. This can be suppressed with the `--skip-triggers` option. (Bug #10431)
- Added new `ER_STACK_OVERRUN_NEED_MORE` error message to indicate that, while the stack is not completely full, more stack space is required. (Bug #11213)
- NDB: Improved handling of the configuration variables `NoOfPagesToDiskDuringRestartACC`, `NoOfPagesToDiskAfterRestartACC`, `NoOfPagesToDiskDuringRestartTUP`, and `NoOfPagesToDiskAfterRestartTUP` should result in noticeably faster startup times for MySQL Cluster. (Bug #12149)
- Added support of where clause for queries with `FROM DUAL`. (Bug #11745)
- Added an optimization that avoids key access with `NULL` keys for the `ref` method when used in outer joins. (Bug #12144)
- Maximum size of stored procedures increased from 64k to 4Gb. (Bug #11602)
- Added error message for users who attempt `CREATE TABLE ... LIKE` and specify a non-table in the `LIKE` clause. (Bug #6859)

Bugs fixed:

- DDL statements now are allowed in stored procedures if the procedure is not invoked from a stored function or a trigger. Also fixed problems where a `TEMPORARY` statement created by one stored routine was inaccessible to another routine invoked during the same connection. (Bug #11126)
- Creation of the `mysql` group account failed during the RPM installation. (Bug #12348)
- `big5` strings were not being stored in `FULLTEXT` index. (Bug #12075)
- When `DROP DATABASE` was called concurrently with a `DROP TABLE` of any

table the MySQL Server crashed. (Bug #12212)

- `max_connections_per_hour` setting was being capped by unrelated `max_user_connections` setting. (Bug #9947)
- `SELECT @@local...` returned `@@session...` in the column header. (Bug #10724)
- Multiplying `ABS()` output by a negative number would return incorrect results. (Bug #11402)
- Updated dependency list for RPM builds to include missing dependencies such as `useradd` and `groupadd`. (Bug #12233)
- `mysql_install_db` used static `localhost` value in GRANT tables even when server hostname is not `localhost`, such as `localhost.localdomain`. This change is applied to version 5.0.10b on Windows. (Bug #11822)
- Multiple `SELECT SQL_CACHE` queries in a stored procedure causes error and client hang. (Bug #6897)
- Added checks to prevent error when allocating memory when there was insufficient memory available. (Bug #7003)
- Character data truncated when GBK characters `0xA3A0` and `0xA1` are present. (Bug #11987)
- Comparisons like `SELECT "A\\" LIKE "A\\"`; fail when using `SET NAMES utf8;`. (Bug #11754)
- When used in a `SELECT` query against a view, the `GROUP_CONCAT()` function returned only a single row. (Bug #11412)
- Calling the C API function `mysql_stmt_fetch()` after all rows of a result set were exhausted would return an error instead of `MYSQL_NO_DATA`. (Bug #11037)
- Information about a trigger was not displayed in the output of `SELECT ... FROM INFORMATION_SCHEMA.TRIGGERS` when the selected database was `INFORMATION_SCHEMA`, prior to the trigger's first invocation. (Bug #12127)

- Issuing successive `FLUSH TABLES WITH READ LOCK` would cause the `mysql` client to hang. (Bug #11934)
- In stored procedures, a cursor that fetched an empty string into a variable would set the variable to `NULL` instead. (Bug #8692)
- A trigger dependent on a feature of one `SQL_MODE` setting would cause an error when invoked after the `SQL_MODE` was changed. (Bug #5891)
- A delayed insert that would duplicate an existing record crashed the server instead. (Bug #12226)
- `ALTER TABLE` when `SQL_MODE = 'TRADITIONAL'` gave rise to an invalid error message. (Bug #11964)
- Attempting to repair a table having a fulltext index on a column containing words whose length exceeded 21 characters and where `myisam_repair_threads` was greater than 1 would crash the server. (Bug #11684)
- The MySQL Cluster backup log was invalid where the number of Cluster nodes was not equal to a power of 2. (Bug #11675)
- `GROUP_CONCAT()` sometimes returned a result with a different collation from that of its arguments. (Bug #10201)
- The `LPAD()` and `RPAD()` functions returned the wrong length to `mysql_fetch_fields()`. (Bug #11311)
- A `UNIQUE VARCHAR` column would be mis-identified as `MUL` in table descriptions. (Bug #11227)
- Incorrect error message displayed if user attempted to create a table in a non-existing database using `CREATE database_name.table_name` syntax. (Bug #10407)
- InnoDB: Do not flush after each write, not even before setting up the doublewrite buffer. Flushing can be extremely slow on some systems. (Bug #12125)

- InnoDB: True VARCHAR: Return NULL columns in the format expected by MySQL. (Bug #12186)
- Two threads could potentially initialize different character sets and overwrite each other. (Bug #12109)
- Unsigned LONG system variables may return incorrect value when retrieved with a SELECT for certain values. (Bug #10351)
- Prepared statements were not being written to the Slow Query log. (Bug #9968)

D.1.17. Changes in release 5.0.10 (27 July 2005)

Functionality added or changed:

- Security improvement: Applied a patch that addresses a zlib data vulnerability that could result in a buffer overflow and code execution. ([CVE-2005-2096](#)) (Bug #11844)
- **Incompatible change:** The namespace for triggers has changed. Previously, trigger names had to be unique per table. Now they must be unique within the schema (database). An implication of this change is that DROP TRIGGER syntax now uses a schema name instead of a table name (schema name is optional and, if omitted, the current schema will be used). (Bug #5892)

Note: When upgrading from a previous version of MySQL 5 to MySQL 5.0.10 or newer, you must drop all triggers and re-create them or DROP TRIGGER will not work after the upgrade. A suggested procedure for doing this is given in [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#).

- The viewing of triggers and trigger metadata has been enhanced as follows:
 - An extension to the SHOW command has been added: SHOW TRIGGERS can be used to view a listing of triggers. See [Section 13.5.4.23, “SHOW TRIGGERS Syntax”](#), for details.
 - The INFORMATION_SCHEMA database now includes a TRIGGERS table. See [Section 20.16, “The INFORMATION_SCHEMA TRIGGERS Table”](#), for

details. (Bug #9586)

- Triggers can now reference tables by name. See [Section 18.1, “CREATE TRIGGER Syntax”](#), for more information.
- The output of `perro - -help` now displays the `- -ndb` option. (Bug #11999)
- On Windows, the search path used by MySQL applications for `my.ini` now includes `.\my.ini` (that is, the application's parent directory, and hence, the installation directory). (Bug #10419)
- Add the `- -defaults-group-suffix` option. See [Section 4.3.2, “Using Option Files”](#).
- Added `mysql_get_character_set_info()` C API function for obtaining information about the default character set of the current connection.
- The bundled version of the `readline` library was upgraded to version 5.0.
- It is no longer necessary to issue an explicit `LOCK TABLES` for any tables accessed by a trigger prior to executing any statements that might invoke the trigger. (Bug #9581, Bug #8406)
- MySQL Cluster: A new `-P` option is available for use with the **ndb_mgmd** client. When called with this option, **ndb_mgmd** prints all configuration data to `stdout`, then exits.
- Add `table_lock_wait_timeout` global server system variable.

Bugs fixed:

- NDB: Trying to use a greater number of tables than specified by the value of `MaxNumberOfTables` caused table corruption such that data nodes could not be restarted. (Bug #9994)
- NDB: Attempting to create or drop tables during a backup would cause the cluster to shut down. (Bug #11942)
- When attempting to drop a table with a broken unique index, NDB failed to drop the table and erroneously report that the table was unknown. (Bug

#11355)

- `SELECT ... NOT IN()` gave unexpected results when only static value present between the `()`. (Bug #11885)
- Fixed compile error when using GCC4 on AMD64. (Bug #12040)
- NDB ignored the `Hostname` option in the `NDBD DEFAULT` section of the Cluster configuration file. (Bug #12028)
- `SHOW PROCEDURE/FUNCTION STATUS` didn't work for users with limited access. (Bug #11577)
- MySQL server would crash is a fetch was performed after a `ROLLBACK` when cursors were involved. (Bug #10760)
- The temporary tables created by an `ALTER TABLE` on a cluster table were visible to all MySQL servers. (Bug #12055)
- `NDB_MGMD` was leaking file descriptors. (Bug #11898)
- IP addresses not shown in `ndb_mgm SHOW` command on second `ndb_mgmd` (or on `ndb_mgmd` restart). (Bug #11596)
- Functions that evaluate to constants (such as `NOW()` and `CURRENT_USER()`) were being evaluated in the definition of a `VIEW` rather than included verbatim. (Bug #4663)
- Execution of `SHOW TABLES` failed to increment the `Com_show_tables` status variable. (Bug #11685)
- For execution of a stored procedure that refers to a view, changes to the view definition were not seen. The procedure continued to see the old contents of the view. (Bug #6120)
- For prepared statements, the SQL parser did not disallow `'?` parameter markers immediately adjacent to other tokens, which could result in malformed statements in the binary log. (For example, `SELECT * FROM t WHERE? = 1` could become `SELECT * FROM t WHERE0 = 1.`) (Bug #11299)

- When two threads compete for the same table, a deadlock could occur if one thread has also a lock on another table through `LOCK TABLES` and the thread is attempting to remove the table in some manner and the other thread want locks on both tables. (Bug #10600)
- Aliasing the column names in a `VIEW` did not work when executing a `SELECT` query on the `VIEW`. (Bug #11399)
- Performing an `ORDER BY` on a `SELECT` from a `VIEW` produced unexpected results when `VIEW` and underlying table had the same column name on different columns. Bug #11709)
- The C API function `mysql_statement_reset()` did not clear error information. (Bug #11183)
- When used within a subquery, `SUBSTRING()` returned an empty string. (Bug #10269)
- Multiple-table `UPDATE` queries using `CONVERT_TZ()` would fail with an error. (Bug #9979)
- `mysql_fetch_fields()` returned incorrect length information for `MEDIUM` and `LONG TEXT` and `BLOB` columns. (Bug #9735)
- `mysqlbinlog` was failing the test suite on Windows due to `BOOL` being incorrectly cast to `INT`. (Bug #11567)
- `NDBCluster`: Server left core files following shutdown if data nodes had failed. (Bug #11516)
- Creating a trigger in one database that references a table in another database was being allowed without generating errors. (Bug #8751)
- Duplicate trigger names were allowed within a single schema. (Bug #6182)
- Server did not accept some fully-qualified trigger names. (Bug #8758)
- The traditional SQL mode accepted invalid dates if the date value provided was the result of an implicit type conversion. (Bug #5906)

- The MySQL server had issues with certain combinations of basedir and datadir. (Bug #7249)
- INFORMATION_SCHEMA.COLUMNS had some inaccurate values for some data types. (Bug #11057)
- LIKE pattern matching using prefix index didn't return correct result. (Bug #11650)
- For several character sets, MySQL incorrectly converted the character code for the division sign to the eucjpm character set. (Bug #11717)
- When invoked within a view, SUBTIME() returned incorrect values. (Bug #11760)
- SHOW BINARY LOGS displayed a file size of 0 for all log files but the current one if the files were not located in the data directory. (Bug #12004)
- Server-side prepared statements failed for columns with a character set of ucs2. (Bug #9442)
- References to system variables in an SQL statement prepared with PREPARE were evaluated during EXECUTE to their values at prepare time, not to their values at execution time. (Bug #9359)
- For server shutdown on Windows, error messages of the form Forcing close of thread n user: 'name' were being written to the error log. Now connections are closed more gracefully without generating error messages. (Bug #7403)
- Increased the version number of the libmysqlclient shared library from 14 to 15 because it is binary incompatible with the MySQL 4.1 client library. (Bug #11893)
- A recent optimizer change caused DELETE ... WHERE ... NOT LIKE and DELETE ... WHERE ... NOT BETWEEN to not properly identify the rows to be deleted. (Bug #11853)
- Within a stored procedure that selects from a table, invoking another procedure that requires a write lock for the table caused that procedure to

fail with a message that the table was read-locked. (Bug #9565)

- Within a stored procedure, selecting from a table through a view caused subsequent updates to the table to fail with a message that the table was read-locked. (Bug #9597)
- For a stored procedure defined with SQL SECURITY DEFINER characteristic, CURRENT_USER() incorrectly reported the user invoking the procedure, not the user who defined it. (Bug #7291)
- Creating a table with a SET or ENUM column with the DEFAULT 0 clause caused a server crash if the table's character set was utf8. (Bug #11819)
- With strict SQL mode enabled, ALTER TABLE reported spurious “Invalid default value” messages for columns that had no DEFAULT clause. (Bug #9881)
- In SQL prepared statements, comparisons could fail for values not equally space-padded. For example, SELECT 'a' = 'a '; returns 1, but PREPARE s FROM 'SELECT ?=?'; SET @a = 'a', @b = 'a '; PREPARE s FROM 'SELECT ?=?'; EXECUTE s USING @a, @b; incorrectly returned 0. (Bug #9379)
- Labels in stored routines did not work if the character set was not latin1. (Bug #7088)
- Invoking the DES_ENCRYPT() function could cause a server crash if the server was started without the --des-key-file option. (Bug #11643)
- The server crashed upon execution of a statement that used a stored function indirectly (via a view) if the function was not yet in the connection-specific stored routine cache and the statement would update a Handler_XXX status variable. This fix allows the use of stored routines under LOCK TABLES without explicitly locking the mysql.lock table. However, you cannot use mysql.proc in statements that will combine locking of it with modifications for other tables. (Bug #11554)
- The server crashed when dropping a trigger that invoked a stored procedure, if the procedure was not yet in the connection-specific stored routine cache. (Bug #11889)

- Selecting the result of an aggregate function for an ENUM or SET column within a subquery could result in a server crash. (Bug #11821)
- Incorrect column values could be retrieved from views defined using statements of the form `SELECT * FROM tbl_name`. (Bug #11771)
- The `mysql.proc` table was not being created properly with the proper `utf8` character set and collation, causing server crashes for stored procedure operations if the server was using a multi-byte character set. To take advantage of the bug fix, **mysql_fix_privilege_tables** should be run to correct the structure of the `mysql.proc` table. (Bug #11365)

Note that it is *necessary* to run **mysql_fix_privileges_tables** when upgrading from a previous installation that contains the `mysql.proc` table (that is, from a previous 5.0 installation). Otherwise, creating stored procedures might not work.

- Execution of a prepared statement that invoked a non-existent or dropped stored routine would crash the server. (Bug #11834)
- Executing a statement that invoked a trigger would cause problems unless a `LOCK TABLES` was first issued for any tables accessed by the trigger. **Note:** The exact nature of the problem depended upon the MySQL 5.0 release being used: prior to 5.0.3, this resulted in a crash; from 5.0.3 to 5.0.7, MySQL would issue a warning; in 5.0.9, the server would issue an error. (Bug #8406)

The same issue caused `LOCK TABLES` to fail following `UNLOCK TABLES` if triggers were involved. (Bug #9581)

- In a shared Windows environment, MySQL could not find its configuration file unless the file was in the `c:\` directory. (Bug #5354)

D.1.18. Changes in release 5.0.9 (15 July 2005)

Functionality added or changed:

- An attempt to create a `TIMESTAMP` column with a display width (for example, `TIMESTAMP(6)`) now results in a warning. Display widths have not been supported for `TIMESTAMP` since MySQL 4.1. (Bug #10466)

- InnoDB: When creating or extending an InnoDB data file, at most one megabyte at a time is allocated for initializing the file. Previously, InnoDB allocated and initialized 1 or 8 megabytes of memory, even if only a few 16-kilobyte pages were to be written. This improves the performance of `CREATE TABLE` in `innodb_file_per_table` mode.
- InnoDB: Various optimizations. Removed unreachable debug code from non-debug builds. Added hints for the branch predictor in `gcc`. Made assertions occupy less space.
- InnoDB: Make `innodb_thread_concurrency=20` by default. Bypass the concurrency checking if the setting is greater than or equal to 20.
- InnoDB: Make `CHECK TABLE` killable. (Bug #9730)
- Recursion in stored routines is now disabled because it was crashing the server. We plan to modify stored routines to allow this to operate safely in a future release. (Bug #11394)
- The handling of `BIT` columns has been improved, and should now be much more reliable in a number of cases. (Bug #10617, Bug #11091, Bug #11572)
- `mysql_real_escape_string()` API function now respects `NO_BACKSLASH_ESCAPES` SQL mode. (Bug #10214)

Bugs fixed:

- `SHOW CREATE VIEW` did not take the `ANSI MODE` into account when quoting identifiers. (Bug #6903)
- The `mysql_config` script did not handle symbolic linking properly. (Bug #10986)
- Incorrect results when using `GROUP BY ... WITH ROLLUP` on a `VIEW`. (Bug #11639)
- Instances of the `VAR_SAMP()` function in view definitions were converted to `VARIANCE()`. This is incorrect because `VARIANCE()` is the same as `VAR_POP()`, not `VAR_SAMP()`. (Bug #10651)

- **mysqldump** failed when reloading a view if the view was defined in terms of a different view that had not yet been reloaded. **mysqldump** now creates a dummy table to handle this case. (Bug #10927)
- **mysqldump** could crash for illegal or non-existent table names. (Bug #9358)
- The `--no-data` option for **mysqldump** was being ignored if table names were given after the database name. (Bug #9558)
- The `--master-data` option for **mysqldump** resulted in no error if the binary log was not enabled. Now an error occurs unless the `--force` option is given. (Bug #11678)
- `DES_ENCRYPT()` and `DES_DECRYPT()` require SSL support to be enabled, but were not checking for it. Checking for incorrect arguments or resource exhaustion was also improved for these functions. (Bug #10589)
- When used in joins, `SUBSTRING()` failed to truncate to zero any string values that could not be converted to numbers. (Bug #10124)
- `mysqldump --xml` did not format `NULL` column values correctly. (Bug #9657)
- There was a compression algorithm issue with `myisampack` for very large datasets (where the total size of all records in a single column was on the order of 3 GB or more) on 64-bit platforms. (A fix for other platforms was made in MySQL 5.0.6.) (Bug #8321)
- Temporary tables were created in the data directory instead of `tmpdir`. (Bug #11440)
- MySQL would not compile correctly on QNX due to missing `rint()` function. (Bug #11544)
- A `SELECT DISTINCT col_name` would work correctly with a MyISAM table only when there was an index on `col_name`. (Bug #11484)
- The server would lose table-level `CREATE VIEW` and `SHOW VIEW` privileges following a `FLUSH PRIVILEGES` or server restart. (Bug #9795)

- In strict mode, an INSERT into a view that did not include a value for a NOT NULL column but that did include a WHERE test on the same column would succeed, This happened even though the INSERT should have been prevented due to the failure to supply a value for the NOT NULL column. (Bug #6443)
- Running a CHECK TABLES on multiple views crashed the server. (Bug #11337)
- When a table had a primary key containing a BLOB column, creation of another index failed with the error BLOB/TEXT column used in key specification without keylength, even when the new index did not contain a BLOB column. (Bug #11657)
- NDB Cluster: When trying to open a table that could not be discovered or unpacked, cluster would return error codes which the MySQL server falsely interpreted as operating system errors. (Bug #103651)
- Manually inserting a row with host=' ' into mysql.tables_priv and performing a FLUSH PRIVILEGES would cause the server to crash. (Bug #11330)
- A cursor using a query with a filter on a DATE or DATETIME column would cause the server to crash server after the data was fetched. (Bug #11172)
- Closing a cursor that was already closed would cause MySQL to hang. (Bug #9814)
- Using CONCAT_WS on a column set NOT NULL caused incorrect results when used in a LEFT JOIN. (Bug #11469)
- Signed BIGINT would not accept -9223372036854775808 as a DEFAULT value. (Bug #11215)
- Views did not use indexes on all appropriate queries. (Bug #10031)
- For MEMORY tables, it was possible for updates to be performed using outdated key statistics when the updates involved only very small changes in a very few rows. This resulted in the random failures of queries such as UPDATE t SET col = col + 1 WHERE col_key = 2; where the same

query with no WHERE clause would succeed. (Bug #10178)

- Optimizer performed range check when comparing unsigned integers to negative constants, could cause errors. (Bug #11185)
- Wrong comparison method used in VIEW when relaxed date syntax used (for example, 2005.06.10). (Bug #11325)
- The ENCRYPT() and SUBSTRING_INDEX() functions would cause errors when used with a VIEW. (Bug #7024)
- Clients would hang following some errors with stored procedures. (Bug #9503)
- Combining cursors and subqueries could cause server crash or memory leaks. (Bug #10736)
- If a prepared statement cursor is opened but not completely fetched, attempting to open a cursor for a second prepared statement will fail. (Bug #10794)

D.1.19. Changes in release 5.0.8 (Not released)

Note: Starting with version 5.0.8, changes for MySQL Cluster can be found in the combined Change History.

Functionality added or changed:

- **Warning: Incompatible change:** Previously, conversion of DATETIME values to numeric form by adding zero produced a result in YYYYMMDDHHMMSS format. The result of DATETIME+0 is now in YYYYMMDDHHMMSS.000000 format. (Bug#12268)
- MEMORY tables now support indexes of up to 500 bytes. See [Section 14.4, “The MEMORY \(HEAP\) Storage Engine”](#). (Bug #10566)
- New SQL_MODE - NO_ENGINE_SUBSTITUTION Prevents automatic substitution of storage engine when the requested storage engine is disabled or not compiled in. (Bug #6877)

- The statements CREATE TABLE, TRUNCATE TABLE, DROP DATABASE, and CREATE DATABASE cause an implicit commit. (Bug #6883)
- Expanded on information provided in general log and slow query log for prepared statements. (Bug #8367, Bug #9334)
- Where a GROUP BY query uses a grouping column from the query's SELECT clause, MySQL now issues a warning. This is done because the SQL standard states that any grouping column must unambiguously reference a column of the table resulting from the query's FROM clause, and allowing columns from the SELECT clause to be used as grouping columns is a MySQL extension to the standard.

By way of example, consider the following table:

```
CREATE TABLE users (
  userid INT NOT NULL PRIMARY KEY,
  username VARCHAR(25),
  usergroupid INT NOT NULL
);
```

MySQL allows you to use the alias in this query:

```
SELECT usergroupid AS id, COUNT(userid) AS number_of_users
FROM users
GROUP BY id;
```

However, the SQL standard requires that the column name be used, as shown here:

```
SELECT usergroupid AS id, COUNT(userid) AS number_of_users
FROM users
GROUP BY usergroupid;
```

Queries such as the first of the two shown above will continue to be supported in MySQL; however, beginning with MySQL 5.0.8, using a column alias in this fashion will generate a warning. Note that in the event of a collision between column names and/or aliases used in joins, MySQL attempts to resolve the conflict by giving preference to columns arising from tables named in the query's FROM clause. (Bug #11211)

- The granting or revocation of privileges on a stored routine is no longer

performed when running the server with `--skip-grant-tables` even after the statement `SET @@global.automatic_sp_privileges=1;` has been executed. (Bug #9993)

- Added support for `B'10'` syntax for bit literal. (Bug #10650)

Bugs fixed:

- **Security fix:** On Windows systems, a user with any of the following privileges
 - REFERENCES
 - CREATE TEMPORARY TABLES
 - GRANT OPTION
 - CREATE
 - SELECT

on `*.*` could crash **mysqld** by issuing a `USE LPT1;` or `USE PRN;` command. In addition, any of the commands `USE NUL;`, `USE CON;`, `USE COM1;`, or `USE AUX;` would report success even though the database was not in fact changed. **Note:** Although this bug was thought to be fixed previously, it was later discovered to be present in the MySQL 5.0.7-beta release for Windows. (Bug #9148, [CVE-2005-0799](#))

- A `CREATE TABLE db_name.tbl_name LIKE ...` statement would crash the server when no database was selected. (Bug #11028)
- `SELECT DISTINCT` queries or `GROUP BY` queries without `MIN()` or `MAX()` could return inconsistent results for indexed columns. (Bug #11044)
- The `SHOW INSTANCE OPTIONS` command in MySQL Instance Manager displayed option values incorrectly for options for which no value had been given. (Bug #11200)
- An outer join with an empty derived table (a result from a subquery) returned no result. (Bug #11284)

- An outer join with an ON condition that evaluated to false could return an incorrect result. (Bug #11285)
- `mysqld_safe` would sometimes fail to remove the pid file for the old `mysql` process after a crash. As a result, the server would fail to start due to a false `A mysqld process already exists...` error. (Bug #11122)
- `CAST(... AS DECIMAL)` didn't work for strings. (Bug #11283)
- `NULLIF()` function could produce incorrect results if first argument is `NULL`. (Bug #11142)
- Setting `@@SQL_MODE = NULL` caused an erroneous error message. (Bug #10732)
- Converting a `VARCHAR` column having an index to a different type (such as `TINYTEXT`) gave rise to an incorrect error message. (Bug #10543)

Note that this bugfix induces a slight change in the behavior of indexes: If an index is defined to be the same length as a field (or is left to default to that field's length), and the length of the field is later changed, then the index will adopt the new length of the field. Previously, the size of the index did not change for some field types (such as `VARCHAR`) when the field type was changed.

- `sql_data_access` column of `routines` table of `INFORMATION_SCHEMA` was empty. (Bug #11055)
- A `CAST()` value could not be included in a `VIEW`. (Bug #11387)
- Server crashed when using `GROUP BY` on the result of a `DIV` operation on a `DATETIME` value. (Bug #11385)
- Possible `NULL` values in `BLOB` columns could crash the server when a `BLOB` was used in a `GROUP BY` query. (Bug #11295)
- Fixed 64 bit compiler warning for packet length in replication. (Bug #11064)
- Multiple range accesses in a subquery cause server crash. (Bug #11487)

- An issue with index merging could cause suboptimal index merge plans to be chosen when searching by indexes created on DATE columns. The same issue caused the InnoDB storage engine to issue the warning using a partial-field key prefix in search. (Bug #8441)
- The mysqlhotcopy script was not parsing the output of SHOW SLAVE STATUS correctly when called with the --record_log_pos option. (Bug #7967)
- SELECT * FROM table returned incorrect results when called from a stored procedure, where table had a primary key. (Bug #10136)
- When used in defining a view, the TIME_FORMAT() function failed with calculated values, for example, when passed the value returned by SEC_TO_TIME(). (Bug #7521)
- SELECT DISTINCT ... GROUP BY constant returned multiple rows (it should return a single row). (Bug #8614)
- INSERT INTO SELECT FROM view produced incorrect result when using ORDER BY. (Bug #11298)
- Fixed hang/crash with Boolean full-text search where a query contained more query terms than one-third of the query length (it could be achieved with truncation operator: 'a*b*c*d*'). (Bug #7858)
- Fixed column name generation in VIEW creation to ensure there are no duplicate column names. (Bug #7448)
- An ORDER BY clause sometimes had no effect on the ordering of a result when selecting specific columns (as opposed to using SELECT *) from a view. (Bug #7422)
- Some data definition statements (CREATE TABLE where the table was not a temporary table, TRUNCATE TABLE, DROP DATABASE, and CREATE DATABASE) were not being written to the binary log after a ROLLBACK. This also caused problems with replication. (Bug #6883)
- Calling a stored procedure that made use of an INSERT ... SELECT ... UNION SELECT ... query caused a server crash. (Bug #11060)

- Selecting from a view defined using `SELECT SUM(DISTINCT ...)` caused an error; attempting to execute a `SELECT * FROM INFORMATION_SCHEMA.TABLES` query after defining such a view crashed the server. (Bug #7015)
- The **mysql** client would output a prompt twice following input of very long strings, because it incorrectly assumed that a call to the `_cgets()` function would clear the input buffer. (Bug #10840)
- A three byte buffer overflow in the client functions caused improper exiting of the client when reading a command from the user. (Bug #10841)
- Fixed a problem where a stored procedure caused a server crash if the query cache was enabled. (Bug #9715)
- `SHOW CREATE DATABASE INFORMATION_SCHEMA` returned an “unknown database” error. (Bug #9434)
- Corrected a problem with `IFNULL()` returning an incorrect result on 64-bit systems. (Bug #11235)
- Fixed a problem resolving table names with `lower_case_table_names=2` when the table name lettercase differed in the `FROM` and `WHERE` clauses. (Bug #9500)
- Fixed server crash due to some internal functions not taking into account that for multi-byte character sets, `CHAR` columns could exceed 255 bytes and `VARCHAR` columns could exceed 65,535 bytes. (Bug #11167)
- Fixed locking problems for multiple-statement `DELETE` statements performed within a stored routine, such as incorrectly locking a to-be-modified table with a read lock rather than a write lock. (Bug #11158)
- Fixed a portability problem testing for `crypt()` support that caused compilation problems when using `OpenSSL/yaSSL` on `HP-UX` and `Mac OS X`. (Bug #10675, Bug #11150)
- The hostname cache was not working. (Bug #10931)
- On Windows, `mysqlshow` did not interpret wildcard characters properly if

they were given in the table name argument. (Bug #10947)

- The default hostname for MySQL server was always `mysql`. (Bug #11174)
- Using `PREPARE` to prepare a statement that invoked a stored routine that deallocated the prepared statement caused a server crash. This is prevented by disabling dynamic SQL within stored routines. (Bug #10975) (Note: This restriction was lifted in 5.0.13 for stored procedures, but not stored functions or triggers.)
- Using `PREPARE` to prepare a statement that invoked a stored routine that executed the prepared statement caused a `Packets out of order` error the second time the routine was invoked. This is prevented by disabling dynamic SQL within stored routines. (Bug #7115) (Note: This restriction was lifted in 5.0.13 for stored procedures, but not stored functions or triggers.)
- Using prepared statements within a stored routine (`PREPARE`, `EXECUTE`, `DEALLOCATE`) could cause the client connection to be dropped after the routine returned. This is prevented by disabling dynamic SQL within stored routines. (Bug #10605) (Note: This restriction was lifted in 5.0.13 for stored procedures, but not stored functions or triggers.)
- When using a cursor with a prepared statement, the first execution returned the correct result but was not cleaned up properly, causing subsequent executions to return incorrect results. (Bug #10729)
- MySQL Cluster: Connections between data nodes and management nodes were not being closed following shutdown of `ndb_mgmd`. (Bug #11132)
- MySQL Cluster: **mysqld** processes would not reconnect to cluster following restart of `ndb_mgmd`. (Bug #11221)
- MySQL Cluster: Fixed problem whereby data nodes would fail to restart on 64-bit Solaris (Bug #9025)
- MySQL Cluster: Calling `ndb_select_count()` crashed the cluster when running on Red Hat Enterprise 4/64-bit/Opteron. (Bug #10058)
- MySQL Cluster: Insert records were incorrectly applied by `ndb_restore`,

thus making restoration from backup inconsistent if the binlog contained inserts. (Bug #11166)

- MySQL Cluster: Cluster would time out and crash after first query on 64-bit Solaris 9. (Bug #8918)
- MySQL Cluster: `ndb_mgm` client show command displayed incorrect output after master data node failure. (Bug #11050)
- MySQL Cluster: A delete performed as part of a transaction caused an erroneous result. (Bug #11133)
- MySQL Cluster: Not allowing sufficient parallelism in cluster configuration (for example, `NoOfTransactions` too small) caused `ndb_restore` to fail without providing any error messages. (Bug #10294)
- MySQL Cluster: When using dynamically allocated ports on Linux, cluster would hang on initial startup. (Bug #10893)
- MySQL Cluster: Setting `TransactionInactiveTimeout= 0` did not result in an infinite timeout. (Bug #11290)
- InnoDB: Enforce maximum `CHAR_LENGTH()` of UTF-8 data in `ON UPDATE CASCADE`. (Bug #10409)
- InnoDB: Pad UTF-8 `VARCHAR` columns with `0x20`. Pad UCS2 `CHAR` columns with `0x0020`. (Bug #10511)

D.1.20. Changes in release 5.0.7 (10 June 2005)

Functionality added or changed:

- Security improvement: Applied a patch to fix a UDF library-loading vulnerability that could result in a buffer overflow and code execution. (<http://www.appsecinc.com/resources/alerts/mysql/2005-002.html>)
- Added `mysql_set_character_set()` C API function for setting the default character set of the current connection. This allows clients to affect the character set used by `mysql_real_escape_string()`. (Bug #8317)

- The behavior of the `Last_query_cost` system variable has been changed. The default value is now 0 (rather than -1) and it now has session-level scope (rather than being global). See [Section 5.2.4, “Server Status Variables”](#), for additional information.
- All characters occurring on the same line following the `DELIMITER` keyword will be set as delimiter. For example, `DELIMITER ;;` will set `;;` as the delimiter. This behavior is now consistent between MySQL 5.1 and MySQL 5.0. (Bug #9879)
- The `table`, `type`, and `rows` columns of `EXPLAIN` output can now be `NULL`. This is required for using `EXPLAIN` on `SELECT` queries that use no tables (for example, `EXPLAIN SELECT 1`). (Bug #9899)
- Placeholders now can be used for `LIMIT` in prepared statements. (Bug #7306)
- `SHOW BINARY LOGS` now displays a `File_size` column that indicates the size of each file.
- The `--delayed-insert` option for **`mysqldump`** has been disabled to avoid causing problems with storage engines that do not support `INSERT DELAYED`. (Bug #7815)
- Improved the optimizer to be able to use indexes for expressions of the form `indexed_col NOT IN (val1, val2, ...)` and `indexed_col NOT BETWEEN val1 AND val2..` (Bug #10561)
- Removed `mysqlshutdown.exe` and `mysqlwatch.exe` from the Windows “No Installer” distribution (they had already been removed from the “With Installer” distribution before). Removed those programs from the source distribution.
- Removed `winMySQLAdmin` from the source distribution and from the “No Installer” Windows distribution (it had already been removed from the “With Installer” distribution before).
- InnoDB: In stored procedures and functions, InnoDB no longer takes full explicit table locks for every involved table. Only ‘intention’ locks are taken, similar to those in the execution of an ordinary SQL statement. This

greatly reduces the number of deadlocks.

Bugs fixed:

- **Security update:** A user with limited privileges could obtain information about the privileges of other users by querying objects in the INFORMATION_SCHEMA database for which that user did not have the requisite privileges. (Bug #10964)
- Triggers with dropped functions caused crashes. (Bug #5893)
- Failure of a BEFORE trigger did not prevent the triggering statement from performing its operation on the row for which the trigger error occurred. Now the triggering statement fails as described in [Section 18.3, “Using Triggers”](#). (Bug #10902)
- Issuing a write lock for a table from one client prevented other clients from accessing the table's metadata. For example, if one client issued a LOCK TABLES mydb.mytable WRITE, then a second client attempting to execute a USE mydb; would hang. (Bug #9998)
- The LAST_DAY() failed to return NULL when supplied with an invalid argument. See [Section 12.5, “Date and Time Functions”](#). (Bug #10568)
- The functions COALESCE(), IF(), and IFNULL() performed incorrect conversions of their arguments. (Bug #9939)
- The TIME_FORMAT() function returned incorrect results with some format specifiers. See [Section 12.5, “Date and Time Functions”](#). (Bug #10590)
- Dropping stored routines when the MySQL server had been started with --skip-grant-tables generated extraneous warnings. (Bug #9993)
- A problem with the my_global.h file caused compilation of MySQL to fail on single-processor Linux systems running 2.6 kernels. (Bug #10364)
- The ucs2_turkish_ci collation failed with upper('i'). UPPER/LOWER now can return a string with different length. (Bug #8610)
- OPTIMIZE of InnoDB table does not return 'Table is full' if out of

tablespace. (Bug #8135)

- GROUP BY queries with ROLLUP returned wrong results for expressions containing group by columns. (Bug #7894)
- Fixed bug in FIELD() function where value list contains NULL. (Bug #10944)
- Corrected a problem where an incorrect data type was returned in the result set metadata when using a prepared SELECT DISTINCT statement to select from a view. (Bug #11111)
- Fixed bug in the MySQL Instance manager that caused the version to always be unknown when SHOW INSTANCE STATUS was issued. (Bug #10229)
- Using ORDER BY to sort the results of an IF() that contained a FROM_UNIXTIME() expression returned incorrect results due to integer overflow. (Bug #9669)
- Fixed a server crash resulting from accessing InnoDB tables within stored functions. This is handled by prohibiting statements that do an implicit or explicit commit or rollback within stored functions or triggers. (Bug #10015)
- Fixed a server crash resulting from the second invocation of a stored procedure that selected from a view defined as a join that used ON in the join conditions. (Bug #6866)
- Using ALTER TABLE for a table that had a trigger caused a crash when executing a statement that activated the trigger, and also a crash later with USE db_name for the database containing the table. (Bug #5894)
- Fixed a server crash resulting from an attempt to allocate too much memory when GROUP BY blob_col and COUNT(DISTINCT) were used. (Bug #11088)
- Fixed a portability problem for compiling on Windows with Visual Studio 6. (Bug #11153)
- The incorrect sequence of statements HANDLER tbl_name READ index_name NEXT without a preceding HANDLER tbl_name READ

index_name = (value_list) for an InnoDB table resulted in a server crash rather than an error. (Bug #5373)

- On Windows, with `lower_case_table_names` set to 2, using `ALTER TABLE` to alter a MEMORY or InnoDB table that had a mixed-case name also improperly changed the name to lowercase. (Bug #9660)
- The server timed out SSL connections too quickly on Windows. (Bug #8572)
- Executing `LOAD INDEX INTO CACHE` for a table while other threads were selecting from the table caused a deadlock. (Bug #10602)
- Fixed a server crash resulting from `CREATE TABLE ... SELECT` that selected from a table being altered by `ALTER TABLE`. (Bug #10224)
- The FEDERATED storage engine properly handled outer joins, but not inner joins. (Bug #10848)
- Consistently report `INFORMATION_SCHEMA` table names in uppercase in `SHOW TABLE STATUS` output. (Bug #10059)
- Fixed a failure of `WITH ROLLUP` to sum values properly. (Bug #10982)
- Triggers were not being activated for multiple-table `UPDATE` or `DELETE` statements. (Bug #5860)
- `INSERT BEFORE` triggers were not being activated for `INSERT ... SELECT` statements. (Bug #6812)
- `INSERT BEFORE` triggers were not being activated for implicit inserts (`LOAD DATA`). (Bug #8755)
- If a stored function contained a `FLUSH` statement, the function crashed when invoked. `FLUSH` now is disallowed within stored functions. (Bug #8409)
- Multiple-row `REPLACE` could fail on a duplicate-key error when having one `AUTO_INCREMENT` key and one unique key. (Bug #11080)
- Fixed a server crash resulting from invalid string pointer when inserting

into the `mysql.host` table. (Bug #10181)

- Multiple-table `DELETE` did always delete on the fly from the first table that was to be deleted from. In some cases, when using many tables and it was necessary to access the same row twice in the first table, we could miss some rows-to-be-deleted from other tables. This is now fixed.
- The `mysql_next_result()` function could hang if you were executing many statements in a `mysql_real_query()` call and one of those statements raised an error. (Bug #9992)
- The combination of `COUNT()`, `DISTINCT`, and `CONCAT()` sometimes triggered a memory deallocation bug on Windows resulting in a server crash. (Bug #9593)
- InnoDB: Do very fast shutdown only if `innodb_fast_shutdown=2`, but wait for threads to exit and release allocated memory if `innodb_fast_shutdown=1`. Starting with MySQL/InnoDB 5.0.5, InnoDB would do brutal shutdown also when `innodb_fast_shutdown=1`. (Bug #9673)
- InnoDB: Fixed InnoDB: Error: `stored_select_lock_type` is 0 inside `::start_stmt()`! in a stored procedure call if `innodb_locks_unsafe_for_binlog` was set in `my.cnf`. (Bug #10746)
- InnoDB: Fixed a duplicate key error that occurred with `REPLACE` in a table with an `AUTO-INC` column. (Bug #11005)
- MySQL would pass an incorrect key length to storage engines for `MIN()`. This could cause warnings `InnoDB: Warning: using a partial-field key prefix in search.` in the `.err` log. (Bug #11039, same as Bug #13218 in MySQL 4.1.15)
- Fixed a server crash for `INSERT` or `UPDATE` when the `WHERE` clause contained a correlated subquery that referred to a column of the table being modified. (Bug #6384)
- Fixed a problem causing an incorrect result for columns that include an aggregate function as part of an expression when `WITH ROLLUP` is added to `GROUP BY`. (Bug #7914)

- Fixed a problem with returning an incorrect result from a view that selected a `COALESCE()` expression from the result of an outer join. (Bug #9938)
- MySQL was adding a `DEFAULT` clause to `ENUM` columns that included no explicit `DEFAULT` and were defined as `NOT NULL`. (This is supposed to happen only for columns that are `NULL`.) (Bug #6267)
- Corrected inappropriate error messages that were displayed when attempting to set the read-only `warning_count` and `error_count` system variables. (Bug #10339)

D.1.21. Changes in release 5.0.6 (26 May 2005)

Functionality added or changed:

- **Incompatible change:** MyISAM and InnoDB tables created with `DECIMAL` columns in MySQL 5.0.3 to 5.0.5 will appear corrupt after an upgrade to MySQL 5.0.6. Dump such tables with **mysqldump** before upgrading, and then reload them after upgrading. (The same incompatibility will occur for these tables created in MySQL 5.0.6 after a downgrade to MySQL 5.0.3 to 5.0.5.) (Bug #10465, Bug #10625)
- **Incompatible change:** The behavior of `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` has changed when the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values both are empty. Formerly, a column was read or written the display width of the column. For example, `INT(4)` was read or written using a field with a width of 4. Now columns are read and written using a field width wide enough to hold all values in the field. However, data files written before this change was made might not be reloaded correctly with `LOAD DATA INFILE` for MySQL 4.1.12 and up. This change also affects data files read by **mysqlimport** and written by **mysqldump --tab**, which use `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`. For more information, see [Section 13.2.5, “LOAD DATA INFILE Syntax”](#). (Bug#12564)
- The precision of the `DECIMAL` data type has been increased from 64 to 65 decimal digits.
- Added the `div_precision_increment` system variable, which indicates the

number of digits of precision by which to increase the result of division operations performed with the / operator.

- Added the `log_bin_trust_routine_creators` system variable, which applies when binary logging is enabled. It controls whether stored routine creators can be trusted not to create stored routines that will cause unsafe events to be written to the binary log.
- Added the `--log-bin-trust-routine-creators` server option for setting the `log_bin_trust_routine_creators` system variable from the command line.
- Implemented the `STMT_ATTR_PREFETCH_ROWS` option for the `mysql_stmt_attr_set()` C API function. This sets how many rows to fetch at a time when using cursors with prepared statements.
- The `GRANT` and `REVOKE` statements now support an `object_type` clause to be used for disambiguating whether the grant object is a table, a stored function, or a stored procedure. Use of this clause requires that you upgrade your grant tables. See [Section 5.6.1, “mysql_fix_privilege_tables — Upgrade MySQL System Tables”](#). (Bug #10246)
- Added `REFERENCED_TABLE_SCHEMA`, `REFERENCED_TABLE_NAME`, and `REFERENCED_COLUMN_NAME` columns to the `KEY_COLUMN_USAGE` table of `INFORMATION_SCHEMA`. (Bug #9587)
- Added a `--show-warnings` option to **mysql** to cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode. In interactive mode, `\w` and `\W` may be used to enable and disable warning display. (Bug #8684)
- Removed a limitation that prevented use of FIFOs as logging targets (such as for the general query log). This modification *does not apply* to the binary log and the relay log. (Bug #8271)
- Added a `--debug` option to **my_print_defaults**.
- When the server cannot read a table because it cannot read the `.frm` file, print a message that the table was created with a different version of MySQL. (This can happen if you create tables that use new features and

then downgrade to an older version of MySQL.) (Bug #10435)

- `SHOW VARIABLES` now shows the `slave_compressed_protocol`, `slave_load_tmpdir` and `slave_skip_errors` system variables. (Bug #7800)
- Removed unused system variable `myisam_max_extra_sort_file_size`.
- Changed default value of `myisam_data_pointer_size` from 4 to 6. This allows us to avoid `table is full` errors for most cases.
- The variable `concurrent_insert` now takes 3 values. Setting this to 2 changes MyISAM to do concurrent inserts to end of table if table is in use by another thread.
- New `/*>` prompt for **mysql**. This prompt indicates that a `/* ... */` comment was begun on an earlier line and the closing `*/` sequence has not yet been seen. (Bug #9186)
- If strict SQL mode is enabled, `VARCHAR` and `VARBINARY` columns with a length greater than 65,535 no longer are silently converted to `TEXT` or `BLOB` columns. Instead, an error occurs. (Bug #8295, Bug #8296)
- The `INFORMATION_SCHEMA.SCHEMATA` table now has a `DEFAULT_COLLATION_NAME` column. (Bug #8998)
- InnoDB: When the maximum length of `SHOW INNODB STATUS` output would be exceeded, truncate the beginning of the list of active transactions, instead of truncating the end of the output. (Bug #5436)
- InnoDB: If `innodb_locks_unsafe_for_binlog` option is set and the isolation level of the transaction is not set to serializable then InnoDB uses a consistent read for select in clauses like `INSERT INTO ... SELECT` and `UPDATE ... (SELECT)` that do not specify `FOR UPDATE` or `IN SHARE MODE`. Thus no locks are set to rows read from selected table.
- Updated version of `libedit` to 2.9. (Bug #2596)
- Removed `mysqlshutdown.exe` and `mysqlwatch.exe` from the Windows “With Installer” distribution.

Bugs fixed:

- An error in the implementation of the MyISAM compression algorithm caused `myisampack` to fail with very large sets of data (total size of all the records in a single column needed to be ≥ 3 GB in order to trigger this issue). (Bug #8321)
- Statements that create and use stored routines were not being written to the binary log, which affects replication and data recovery options. (Bug #2610) Stored routine-related statements now are logged, subject to the issues and limitations discussed in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).
- Disabled binary logging within stored routines to avoid writing spurious extra statements to the binary log. For example, if a routine `p()` executes an `INSERT` statement, then for `CALL p()`, the `CALL` statement appears in the binary log, but not the `INSERT` statement. (Bug #9100)
- Statements that create and drop triggers were not being written to the binary log, which affects replication and data recovery options. (Bug #10417) Trigger-related statements now are logged, subject to the issues and limitations discussed in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).
- The `mysql_stmt_execute()` and `mysql_stmt_reset()` C API functions now close any cursor that is open for the statement, which prevents a server crash. (Bug #9478)
- The `mysql_stmt_attr_set()` C API function now returns an error for option values that are defined in `mysql.h` but not yet implemented, such as `CURSOR_TYPE_SCROLLABLE`. (Bug #9643)
- `MERGE` tables could fail on Windows due to incorrect interpretation of pathname separator characters for filenames in the `.MRG` file. (Bug #10687)
- Fixed a server crash for `INSERT ... ON DUPLICATE KEY UPDATE` with `MERGE` tables, which do not have unique indexes. (Bug #10400)
- Fix `FORMAT()` to do better rounding for double values (for example, `FORMAT(4.55, 1)` returns 4.6, not 4.5). (Bug #9060)

- Disallow use of `SESSION` or `GLOBAL` for user variables or local variables in stored routines. (Bug #9286)
- Fixed a server crash when using `GROUP BY ... WITH ROLLUP` on an indexed column in an InnoDB table. (Bug #9798)
- In strict SQL mode, some assignments to numeric columns that should have been rejected were not (such as the result of an arithmetic expression or an explicit `CAST()` operation). (Bug #6961)
- `CREATE TABLE t AS SELECT UUID()` created a `VARCHAR(12)` column, which is too small to hold the 36-character result from `UUID()`. (Bug #9535)
- Fixed a server crash in the `BLACKHOLE` storage engine. (Bug #10175)
- Fixed a server crash resulting from repeated calls to `ABS()` when the argument evaluated to `NULL`. (Bug #10599)
- For a user-defined function invoked from within a prepared statement, the UDF's initialization routine was invoked for each execution of the statement, but the deinitialization routine was not. (It was invoked only when the statement was closed.) Similarly, when invoking a UDF from within a trigger, the initialization routine was invoked but the deinitialization routine was not. For UDFs that have an expensive deinit function (such as `myperl`, this bugfix will have negative performance consequences. (Bug #9913)
- Portability fix for Cygwin: Don't use `#pragma interface` in source files. (Bug #10241)
- Fix `CREATE TABLE ... LIKE` to work when `lower_case_table_names` is set on a case-sensitive filesystem and the source table name is not given in lowercase. (Bug #9761)
- Fixed a server crash resulting from a `CHECK TABLE` statement where the arguments were a view name followed by a table name. (Bug #9897)
- Within a stored procedure, attempting to update a view defined as an inner join failed with a `Table 'tbl_name' was locked with a READ lock and can't be updated` error. (Bug #9481)

- Fixed a problem with INFORMATION_SCHEMA tables being inaccessible depending on lettercase used to refer to them. (Bug #10018)
- **my_print_defaults** was ignoring the --defaults-extra-file option or crashing when the option was given. (Bug #9136, Bug #9851)
- The INFORMATION_SCHEMA.COLUMNS table was missing columns of views for which the user has access. (Bug #9838)
- Fixed a **mysqldump** crash that occurred with the --complete-insert option when dumping tables with a large number of long column names. (Bug #10286)
- Corrected a problem where DEFAULT values were not assigned properly to BIT(1) or CHAR(1) columns if certain other columns preceded them in the table definition. (Bug #10179)
- For MERGE tables, avoid writing absolute pathnames in the .MRG file for the names of the constituent MyISAM tables so that if the data directory is moved, MERGE tables will not break. For **mysqld**, write just the MyISAM table name if it is in the same database as the MERGE table, and a path relative to the data directory otherwise. For the embedded servers, absolute pathnames may still be used. (Bug #5964)
- Corrected a problem resolving outer column references in correlated subqueries when using the prepared statements. (Bug #10041)
- Corrected the error message for exceeding the MAX_CONNECTIONS_PER_HOUR limit to say max_connections_per_hour instead of max_connections. (Bug #9947)
- Fixed incorrect memory block allocation for the query cache in the embedded server. (Bug #9549)
- Corrected an inability to select from a view within a stored procedure. (Bug #9758)
- Fixed a server crash resulting from use of AVG(DISTINCT) with GROUP BY ... WITH ROLLUP. (Bug #9799)

- Fixed a server crash resulting from use of `DISTINCT AVG()` with `GROUP BY ... WITH ROLLUP`. (Bug #9800)
- Fixed a server crash resulting from use of a `CHAR` or `VARCHAR` column with `MIN()` or `MAX()` and `GROUP BY ... WITH ROLLUP`. (Bug #9820)
- Fixed a server crash resulting from use of `SELECT DISTINCT` with a prepared statement that uses a cursor. (Bug #9520)
- Fixed server crash resulting from multiple calls to a stored procedure that assigned the result of a subquery to a variable or compared it to a value with `IN`. (Bug #5963)
- Selecting from a single-table view defined on multiple-table views caused a server crash. (Bug #8528)
- If the file named by a `--defaults-extra-file` option does not exist or is otherwise inaccessible, an error now occurs. (Bug #5056)
- `net_read_timeout` and `net_write_timeout` were not being respected on Windows. (Bug #9721)
- `SELECT` from `INFORMATION_SCHEMA` tables failed if the statement has a `GROUP BY` clause and an aggregate function in the select list. (Bug #9404)
- Corrected some failures of prepared statements for SQL (`PREPARE` plus `EXECUTE`) to return all rows for some `SELECT` statements. (Bug #9096, Bug #9777)
- Remove extra slashes in `--tmpdir` value (for example, convert `/var//tmp` to `/var/tmp`, because they caused various errors. (Bug #8497)
- Added `Create_routine_priv`, `Alter_routine_priv`, and `Execute_priv` privileges to the `mysql.host` privilege table. (They had been added to `mysql.db` in MySQL 5.0.3 but not to the host table.) (Bug #8166)
- Fixed **configure** to properly recognize whether NTPL is available on Linux. (Bug #2173)
- Incomplete results were returned from `INFORMATION_SCHEMA.COLUMNS` for

INFORMATION_SCHEMA tables for non-root users. (Bug #10261)

- Fixed a portability problem in compiling `mysql.cc` with **VC++** on Windows. (Bug #10245)
- `SELECT 0/0` returned 0 rather than NULL. (Bug #10404)
- `MAX()` for an `INT UNSIGNED` (unsigned 4-byte integer) column could return negative values if the column contained values larger than 2^{31} . (Bug #9298)
- `SHOW CREATE VIEW` got confused and could not find the view if there was a temporary table with the same name as the view. (Bug #8921)
- Fixed a deadlock resulting from use of `FLUSH TABLES WITH READ LOCK` while an `INSERT DELAYED` statement is in progress. (Bug #7823)
- The optimizer was choosing suboptimal execution plans for certain outer joins where the right table of a left join (or left table of a right join) had both `ON` and `WHERE` conditions. (Bug #10162)
- `RENAME TABLE` for an `ARCHIVE` table failed if the `.arn` file was not present. (Bug #9911)
- Invoking a stored function that executed a `SHOW` statement resulted in a server crash. (Bug #8408)
- Fixed problems with static variables and do not link with `libsupc++` to allow building on FreeBSD 5.3. (Bug #9714)
- Fixed some **awk** script portability problems in **cmd-line-utils/libedit/makelist.sh**. (Bug #9954)
- Fixed a problem with mishandling of NULL key parts in hash indexes on `VARCHAR` columns, resulting in incorrect query results. (Bug #9489, Bug #10176)
- InnoDB: Fixed a critical bug in InnoDB `AUTO_INCREMENT`: it could assign the same value for several rows. (Bug #10359) InnoDB: All InnoDB bug fixes from 4.1.12 and earlier versions, and also the fixes to bugs #10335 and #10607 listed in the 4.1.13 change notes.

D.1.22. Changes in release 5.0.5 (Not released)

No public release of MySQL 5.0.5 was made. The changes described in this section are available in MySQL 5.0.6.

Functionality added or changed:

- Added support for the BIT data type to the MEMORY, InnoDB, and BDB storage engines.
- SHOW VARIABLES no longer displays the deprecated log_update system variable. (Bug #9738)
- The behavior controlled by the --innodb-fast-shutdown option now can be changed at runtime by setting the value of the global innodb_fast_shutdown system variable. It now accepts values 0, 1 and 2 (except on Netware where 2 is disabled). If set to 2, then when the MySQL server shuts down, InnoDB will just flush its logs and shut down brutally (and quickly) as if a MySQL crash had occurred; no committed transaction will be lost, but a crash recovery will be done at next startup.

Bugs fixed:

- **Security fix:** If **mysqld** was started with --user=non_existent_user, it would run using the privileges of the account it was invoked from, even if that was root. (Bug #9833)
- Corrected a failure to resolve a column reference correctly for a LEFT JOIN that compared a join column to an IN subquery. (Bug #9338)
- Fixed a problem where, after an internal temporary table in memory became too large and had to be converted to an on-disk table, the error indicator was not cleared and the query failed with error 1023 (Can't find record in ''). (Bug #9703)
- Multiple-table updates could produce spurious data-truncation warnings if they used a join across columns that are indexed using a column prefix. (Bug #9103)
- Fixed a string-length comparison problem that caused **mysqld** to fail loading

dump files containing certain ‘\’-sequences. (Bug #9756)

- Fixed a failure to resolve a column reference properly when an outer join involving a view contained a subquery and the column was used in the subquery and the outer query. (Bug #6106, Bug #6107)
- Use of a subquery that used `WITH ROLLUP` in the `FROM` clause of the main query sometimes resulted in a `Column cannot be null` error. (Bug #9681)
- Fixed a memory leak that occurred when selecting from a view that contained a subquery. (Bug #10107)
- Fixed an optimizer bug in computing the union of two ranges for the `OR` operator. (Bug #9348)
- Fixed a segmentation fault in **mysqlcheck** that occurred when the last table checked in `--auto-repair` mode returned an error (such as the table being a `MERGE` table). (Bug #9492)
- `SET @var= CAST(NULL AS [INTEGER|CHAR])` now sets the result type of the variable to `INTEGER/CHAR`. (Bug #6598)
- Incorrect results were returned for queries of the form `SELECT ... LEFT JOIN ... WHERE EXISTS (subquery)`, where the subquery selected rows based on an `IS NULL` condition. (Bug #9516)
- Executing `LOCK TABLES` and then calling a stored procedure caused an error and resulting in the server thinking that no stored procedures exist. (Bug #9566)
- Selecting from a view containing a subquery caused the server to hang. (Bug #8490)
- Within a stored procedure, attempting to execute a multiple-table `UPDATE` failed with a `Table 'tbl_name' was locked with a READ lock and can't be updated` error. (Bug #9486)
- Starting **mysqld** with the `--skip-innodb` and `--default-storage-engine=innodb` (or `--default-table-type=innodb`) caused a server crash. (Bug #9815)

- Queries containing `CURRENT_USER()` incorrectly were registered in the query cache. (Bug #9796)
- Setting the `storage_engine` system variable to `MEMORY` succeeded, but retrieving the variable resulted in a value of `HEAP` (the old name for the `MEMORY` storage engine) rather than `MEMORY`. (Bug #10039)
- **mysqlshow** displayed an incorrect row count for tables. (Bug #9391)
- The server died with signal 11 if a non-existent location was specified for the location of the binary log. Now the server exits after printing an appropriate error message. (Bug #9542)
- Fixed a problem in the client/server protocol where the server closed the connection before sending the final error message. The problem could show up as a `Lost connection to MySQL server during query` when attempting to connect to access a non-existent database. (Bug #6387, Bug #9455)
- Fixed a `readline`-related crash in **mysql** when the user pressed Control-R. (Bug #9568)
- For stored functions that should return a `YEAR` value, corrected a failure of the value to be in `YEAR` format. (Bug #8861)
- Fixed a server crash resulting from invocation of a stored function that returned a value having an `ENUM` or `SET` data type. (Bug #9775)
- Fixed a server crash resulting from invocation of a stored function that returned a value having a `BLOB` data type. (Bug #9102)
- Fixed a server crash resulting from invocation of a stored function that returned a value having a `BIT` data type. (Bug #7648)
- `TIMEDIFF()` with a negative time first argument and positive time second argument produced incorrect results. (Bug #8068)
- Fixed a problem with `OPTIMIZE TABLE` for `InnoDB` tables being written twice to the binary log. (Bug #9149)

- InnoDB: Prevent ALTER TABLE from changing the storage engine if there are foreign key constraints on the table. (Bug #5574, Bug #5670)
- InnoDB: Fixed a bug where next-key locking doesn't allow the insert which does not produce a phantom. (Bug #9354) If the range is of type 'a' <= uniquecolumn, InnoDB lock only the RECORD, if the record with the column value 'a' exists in a CLUSTERED index. This allows inserts before a range.
- InnoDB: When FOREIGN_KEY_CHECKS=0, ALTER TABLE and RENAME TABLE will ignore any type incompatibilities between referencing and referenced columns. Thus, it will be possible to convert the character sets of columns that participate in a foreign key. Be sure to convert all tables before modifying any data! (Bug #9802)
- Provide more informative error messages in clustered setting when a query is issued against a table that has been modified by another **mysqld** server. (Bug #6762)

D.1.23. Changes in release 5.0.4 (16 April 2005)

Functionality added or changed:

- Added ENGINE=MyISAM table option when creating mysql.proc table in **mysql_create_system_tables** script to make sure the table is created as a MyISAM table even if the default storage engine has been changed. (Bug #9496)
- SHOW CREATE TABLE for an INFORMATION_SCHEMA table no longer prints a MAX_ROWS value because the value has no meaning. (Bug #8941)
- Invalid DEFAULT values for CREATE TABLE now generate errors. (Bug #5902)
- Added --show-table-type option to **mysqlshow**, to display a column indicating the table type, as in SHOW FULL TABLES. (Bug #5036)
- The way the time zone information is stored in the binary log was changed, so that it is now possible to have a replication master and slave running with different global time zones. A drawback is that replication from 5.0.4 masters to pre-5.0.4 slaves is impossible.

- Added `--with-big-tables` compilation option to **configure**. (Previously it was necessary to pass `-DBIG_TABLES` to the compiler manually in order to enable large table support.) See [Section 2.9.2, “Typical configure Options”](#), for details.
- New configuration directives `!include` and `!includedir` implemented for including option files and searching directories for option files. See [Section 4.3.2, “Using Option Files”](#), for usage.

Bugs fixed:

- The use of `XOR` together with `NOT ISNULL()` erroneously resulted in some outer joins being converted to inner joins by the optimizer. (Bug #9017)
- Fixed an optimizer problem where extraneous comparisons between `NULL` values in indexed columns were being done for operators such as `=` that are never true for `NULL`. (Bug #8877)
- Fixed the client/server protocol for prepared statements so that reconnection works properly when the connection is killed while `reconnect` is enabled. (Bug #8866)
- A server installed as a Windows service and started with `--shared-memory` could not be stopped. (Bug #9665)
- Fixed a server crash resulting from multiple executions of a prepared statement involving a join of an `INFORMATION_SCHEMA` table with another table. (Bug #9383)
- Fixed `utf8_spanish2_ci` and `ucs2_spanish2_ci` collations to not consider ‘r’ equal to ‘rr’. If you upgrade to this version from an earlier version, you should rebuild the indexes of affected tables. (Bug #9269)
- **mysqldump** dumped core when invoked with `--tmp` and `--single-transaction` options and a non-existent table name. (Bug #9175)
- Allow extra HKSCS and cp950 characters (big5 extension characters) to be accepted in `big5` columns. (Bug #9357)
- **mysql.server** no longer uses non-portable **alias** command or `LSB`

functions. (Bug #9852)

- Fixed a server crash resulting from `GROUP BY` on a decimal expression. (Bug #9210)
- In prepared statements, subqueries containing parameters were erroneously treated as `const` tables during preparation, resulting in a server crash. (Bug #8807)
- InnoDB: `ENUM` and `SET` columns were treated incorrectly as character strings. This bug did not manifest itself with `latin1` collations if there were less than about 100 elements in an `ENUM`, but it caused malfunction with `UTF-8`. Old tables will continue to work. In new tables, `ENUM` and `SET` will be internally stored as unsigned integers. (Bug #9526)
- InnoDB: Avoid test suite failures caused by a locking conflict between two server instances at server shutdown/startup. This conflict on advisory locks appears to be the result of a bug in the operating system; these locks should be released when the files are closed, but somehow that does not always happen immediately in Linux. (Bug #9381)
- InnoDB: True `VARCHAR`: InnoDB stored the 'position' of a row wrong in a column prefix primary key index; this could cause MySQL to complain `ERROR 1032: Can't find record ...` in an update of the primary key, and also some `ORDER BY` or `DISTINCT` queries. (Bug #9314)
- InnoDB: Fix bug in MySQL/InnoDB 5.0.3: SQL statements were not rolled back on error. (Bug #8650)
- Fixed a `Commands out of sync` error when two prepared statements for single-row result sets were open simultaneously. (Bug #8880)
- Fixed a server crash after a call to `mysql_stmt_close()` for single-row result set. (Bug #9159)
- Fixed server crashes for `CREATE TABLE ... SELECT` or `INSERT INTO ... SELECT` when selecting from multiple-table view. (Bug #8703, Bug #9398)
- `TRADITIONAL SQL` mode should prevent inserts where a column with no default value is omitted or set to a value of `DEFAULT`. Fixed cases where this

restriction was not enforced. (Bug #5986)

- Fixed a server crash when creating a PRIMARY KEY for a table, if the table contained a BIT column. (Bug #9571)
- Warning message from GROUP_CONCAT() did not always indicate correct number of lines. (Bug #8681)
- The commit count cache for NDB was not properly invalidated when deleting a record using a cursor. (Bug #8585)
- Fixed option-parsing code for the embedded server to understand K, M, and G suffixes for the net_buffer_length and max_allowed_packet options. (Bug #9472)
- Selecting a BIT column failed if the binary client/server protocol was used. (Bug #9608)
- Fixed a permissions problem whereby information in INFORMATION_SCHEMA could be exposed to a user with insufficient privileges. (Bug #7214)
- An error now occurs if you try to insert an invalid value via a stored procedure in STRICT mode. (Bug #5907)
- Link with libsupc++ on Fedora Core 3 to get language support functions. (Bug #6554)
- The value of the CHARACTER_MAXIMUM_LENGTH and CHARACTER_OCTET_LENGTH columns of the INFORMATION_SCHEMA.COLUMNS table must be NULL for numeric columns, but were not. (Bug #9344)
- DROP TABLE did not drop triggers that were defined for the table. DROP DATABASE did not drop triggers in the database. (Bug #5859, Bug #6559)
- CREATE OR REPLACE VIEW and ALTER VIEW now require the CREATE VIEW and DROP privileges, not CREATE VIEW and DELETE. (DELETE is a row-level privilege, not a table-level privilege.) (Bug #9260)
- Some user variables were not being handled with “implicit” coercibility. (Bug #9425)

- Setting the `max_error_count` system variable to 0 resulted in a setting of 1. (Bug #9072)
- Fixed a collation coercibility problem that caused a union between binary and non-binary columns to fail. (Bug #6519)
- Fixed a bug in division of floating point numbers. It could cause nine zeros (000000000) to be inserted in the middle of the quotient. (Bug #9501)
- `INFORMATION_SCHEMA` tables had an implicit upper limit for the number of rows. As a result, not all data could be returned for some queries. (Bug #9317)
- Fixed a problem with the `tee` command in **mysql** that resulted in **mysql** crashing. (Bug #8499)
- `CAST()` now produces warnings when casting incorrect `INTEGER` and `CHAR` values. This also applies to implicit string to number casts. (Bug #5912)
- `ALTER TABLE` now fails in `STRICT` mode if the alteration generates warnings.
- Using `CONVERT('0000-00-00', date)` or `CAST('0000-00-00' as date)` with the `NO_ZERO_DATE` SQL mode enabled now produces a warning. (Bug #6145)
- Inserting a zero date in a `DATE`, `DATETIME` or `TIMESTAMP` column during `TRADITIONAL` mode now produces an error. (Bug #5933)
- Inserting a zero date into a `DATETIME` column in `TRADITIONAL` mode now produces an error.
- `STR_TO_DATE()` now produces errors in strict mode (and warnings otherwise) when given an illegal argument. (Bug #5902)
- Fixed a problem with `ORDER BY` that sometimes caused incorrect sorting of `utf8` data. (Bug #9309)
- Fixed server crash resulting from queries that combined `SELECT DISTINCT`, `SUM()`, and `ROLLUP`. (Bug #8615)

- Incorrect results were returned from queries that combined SELECT DISTINCT, GROUP BY , and ROLLUP. (Bug #8616)
- Too many rows were returned from queries that combined ROLLUP and LIMIT if SQL_CALC_FOUND_ROWS was given. (Bug #8617)
- If on replication master a LOAD DATA INFILE is interrupted in the middle (integrity constraint violation, killed connection...), the slave used to skip this LOAD DATA INFILE entirely, thus missing some changes if this command permanently inserted/updated some table records before being interrupted. This is now fixed. (Bug #3247)

D.1.24. Changes in release 5.0.3 (23 March 2005: Beta)

Note: This Beta release, as any other pre-production release, should not be installed on “production” level systems or systems with critical data. It is good practice to back up your data before installing any new version of software. Although MySQL worked very hard to ensure a high level of quality, protect your data by making a backup as you would for any software beta release.

Functionality added or changed:

- Security improvement: The server creates .frm, .MYD, .MYI, .MRG, .ISD, and .ISM table files only if a file with the same name does not already exist. Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and informing us about this issue. ([CVE-2005-0711](#))
- Security improvement: User-defined functions should have at least one symbol defined in addition to the xxx symbol that corresponds to the main xxx() function. These auxiliary symbols correspond to the xxx_init(), xxx_deinit(), xxx_reset(), xxx_clear(), and xxx_add() functions. **mysqld** by default no longer loads UDFs unless they have at least one auxiliary symbol defined in addition to the main symbol. The --allow-suspicious-udfs option controls whether UDFs that have only an xxx symbol can be loaded. By default, the option is off. **mysqld** also checks UDF filenames when it reads them from the mysql.func table and rejects those that contain directory pathname separator characters. (It already checked names as given in CREATE FUNCTION statements.) See [Section 24.2.4.1, “UDF Calling Sequences for Simple Functions”](#),

[Section 24.2.4.2, “UDF Calling Sequences for Aggregate Functions”](#), and [Section 24.2.4.6, “User-Defined Function Security Precautions”](#). Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and informing us about this issue. ([CVE-2005-0709](#), [CVE-2005-0710](#))

- The DECIMAL and NUMERIC data types now are handled with a fixed-point library that allows for precision math handling that results in more accurate results. See [Chapter 21, Precision Math](#).

Warning: Incompatible change: A consequence of the change in handling of the DECIMAL and NUMERIC fixed-point data types is that the server is more strict to follow standard SQL. For example, a data type of DECIMAL(3,1) stores a maximum value of 99.9. Previously, the server allowed larger numbers to be stored. That is, it stored a value such as 100.0 as 100.0. Now the server clips 100.0 to the maximum allowable value of 99.9. If you have tables that were created before MySQL 5.0.3 and that contain floating-point data not strictly legal for the data type, you should alter the data types of those columns. For example:

```
ALTER TABLE tbl_name MODIFY col_name DECIMAL(4,1);
```

- **Incompatible change:** The C API ER_WARN_DATA_TRUNCATED warning symbol was renamed to WARN_DATA_TRUNCATED.
- **InnoDB: Upgrading from 4.1:** The sorting order for end-space in TEXT columns for InnoDB tables has changed. Starting from 5.0.3, InnoDB compares TEXT columns as space-padded at the end. If you have a non-unique index on a TEXT column, you should run CHECK TABLE on it, and run OPTIMIZE TABLE if the check reports errors. If you have a UNIQUE INDEX on a TEXT column, you should rebuild the table with OPTIMIZE TABLE.
- Implemented support for XA transactions. See [Section 13.4.7, “XA Transactions”](#). The implementation make the innodb_safe_binlog system variable obsolete, so it has been removed.
- **mysqlbinlog** now prints a ROLLBACK statement at the end of its output, in case the server crashed while it was in the process of writing the final entry into the last binary log named on the command line. This causes any half-written transaction to be rolled back when the output is executed. The ROLLBACK is harmless if the binary log file was written and closed normally.

- Added the `engine_condition_pushdown` system variable. For NDB, setting this variable to 1 allows processing of some `WHERE` clause conditions to be processed in NDB nodes before rows are sent to the MySQL server, rather than having rows sent to the server for evaluation.
- Additional control over transaction completion was implemented. The `COMMIT` and `ROLLBACK` statements support `AND [NO] CHAIN` and `RELEASE` clauses. There is a new `RELEASE SAVEPOINT` statement. The `completion_type` system variable was added for setting the global and session default completion type.
- A new `CREATE USER` privilege was added.
- `my.cnf` in the compile-time `datadir` (usually `/usr/local/mysql/data/` in the binary tarball distributions) is not being read anymore. The value of the environment variable `MYSQL_HOME` is used instead of the hard-coded path.
- Support for the ISAM storage engine has been removed. If you have ISAM tables, you should convert them before upgrading. See [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#).
- Support for RAID options in MyISAM tables has been removed. If you have tables that use these options, you should convert them before upgrading. See [Section 2.11.2, “Upgrading from MySQL 4.1 to 5.0”](#).
- Added support for `AVG(DISTINCT)`.
- `ONLY_FULL_GROUP_BY` no longer is included in the ANSI composite SQL mode. (Bug #8510)
- **mysqld_safe** will create the directory where the UNIX socket file is to be located if the directory does not exist. This applies only to the last component of the directory pathname. (Bug #8513)
- The coercibility for the return value of functions such as `USER()` or `VERSION()` now is “system constant” rather than “implicit.” This makes these functions more coercible than column values so that comparisons of the two do not result in `Illegal mix of collations` errors. `COERCIBILITY()` was modified to accommodate this new coercibility value. See [Section 12.9.3, “Information Functions”](#).

- User variable coercibility has been changed from “coercible” to “implicit.” That is, user variables have the same coercibility as column values.
- Boolean full-text phrase searching now requires only that matches contain exactly the same words as the phrase and in the same order. Non-word characters no longer need match exactly.
- `CHECKSUM TABLE` returns a warning for non-existing tables. The checksum value remains `NULL` as before. (Bug #8256)
- The server now includes a timestamp in the `Ready for connections` message that is written to the error log at startup. (Bug #8444)
- Added `SQL_NOTES` session variable to cause Note-level warnings not to be recorded. (Bug #6662)
- Allowed the service-installation command for Windows servers to specify a single option other than `--defaults-file` following the service name. This is for compatibility with MySQL 4.1. (Bug #7856)
- InnoDB: Commit after every 10,000 copied rows when executing `ALTER TABLE`, `CREATE INDEX`, `DROP INDEX` or `OPTIMIZE TABLE`. This makes it much faster to recover from an aborted operation.
- Added `VAR_POP()` and `STDDEV_POP()` as standard SQL aliases for the `VARIANCE()` and `STDDEV()` functions that compute population variance and standard deviation. Added new `VAR_SAMP()` and `STDDEV_SAMP()` functions to compute sample variance and standard deviation. (Bug #3190)
- Fixed a problem with out-of-order packets being sent (`ERROR` after `OK` or `EOF`) following a `KILL QUERY` statement. (Bug #6804)
- Retrieving from a view defined as a `SELECT` that mixed `UNION ALL` and `UNION DISTINCT` resulted in a different result than retrieving from the original `SELECT`. (Bug #6565)
- Fixed a problem with non-optimal `index_merge` query execution plans being chosen on IRIX. (Bug #8578)
- `BIT` in column definitions now is a distinct data type; it no longer is treated

as a synonym for TINYINT(1).

- Bit-field values can be written using b'value' notation. *value* is a binary value written using 0s and 1s.
- From the Windows distribution, predefined accounts without passwords for remote users ("root@%", "@%") were removed (other distributions never had them).
- Added `mysql_library_init()` and `mysql_library_end()` as synonyms for the `mysql_server_init()` and `mysql_server_end()` C API functions. `mysql_library_init()` and `mysql_library_end()` are `#define` symbols, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL C API library no matter whether the application uses `libmysqlclient` or `libmysqld`. (Bug #6149)
- `SHOW COLUMNS` now displays `NO` rather than blank in the `NULL` output column if the corresponding table column cannot be `NULL`.
- Changed XML format for **mysql** from `<col_name>col_value</col_name>` to `<field name="col_name">col_value</field>` to allow for proper encoding of column names that are not legal as element names. (Bug #7811)
- Added `--innodb-checksums` and `--innodb-doublewrite` options for **mysqld**.
- Added `--large-pages` option for **mysqld**.
- Added `multi_read_range` system variable.
- `SHOW DATABASES`, `SHOW TABLES`, `SHOW COLUMNS`, and so forth display information about the `INFORMATION_SCHEMA` database. Also, several `SHOW` statements now accept a `WHERE` clause specifying which output rows to display. See [Chapter 20, The INFORMATION_SCHEMA Database](#).
- Added the `CREATE ROUTINE` and `ALTER ROUTINE` privileges, and made the `EXECUTE` privilege operational.
- InnoDB: Corrected a bug in the crash recovery of `ROW_FORMAT=COMPACT`

tables that caused corruption. (Bug #7973) There may still be bugs in the crash recovery, especially in COMPACT tables.

- When the MyISAM storage engine detects corruption of a MyISAM table, a message describing the problem now is written to the error log.
- InnoDB: When MySQL/InnoDB is compiled on Mac OS X 10.2 or earlier, detect the operating system version at run time and use the `fcntl()` file flush method on Mac OS X versions 10.3 and later. In Mac OS X, `fsync()` does not flush the write cache in the disk drive, but the special `fcntl()` does; however, the flush request is ignored by some external devices. Failure to flush the buffers may cause severe database corruption at power outages.
- InnoDB: Implemented fast TRUNCATE TABLE. The old approach (deleting rows one by one) may be used if the table is being referenced by foreign keys. (Bug #7150)
- Added cp932 (SJIS for Windows Japanese) and eucjpms (UJIS for Windows Japanese) character sets.
- Added several InnoDB status variables. See [Section 5.2.4, “Server Status Variables”](#).
- Added the FEDERATED storage engine. See [Section 14.7, “The FEDERATED Storage Engine”](#).
- SHOW CREATE TABLE NOW uses USING index_type rather than TYPE index_type to specify an index type. (Bug #7233)
- InnoDB now supports a fast TRUNCATE TABLE. One visible change from this is that auto-increment values for this table are reset on TRUNCATE.
- Added an error member to the MYSQL_BIND data structure that is used in the C API for prepared statements. This member is used for reporting data truncation errors. Truncation reporting is enabled via the new MYSQL_REPORT_DATA_TRUNCATION option for the mysql_options() C API function.
- API change: the reconnect flag in the MYSQL structure is now set to 0 by

`mysql_real_connect()`. Only those client programs which didn't explicitly set this flag to 0 or 1 after `mysql_real_connect()` experience a change. Having automatic reconnection enabled by default was considered too dangerous (after reconnection, table locks, temporary tables, user and session variables are lost).

- `FLUSH TABLES WITH READ LOCK` is now killable while it's waiting for running `COMMIT` statements to finish.
- `MEMORY (HEAP)` can have `VARCHAR()` fields.
- `VARCHAR` columns now remember end space. A `VARCHAR()` column can now contain up to 65535 bytes. For more details, see [Section D.1, “Changes in release 5.0.x \(Production\)”](#). If the table handler doesn't support the new `VARCHAR` type, then it's converted to a `CHAR` column. Currently this happens for NDB tables.
- InnoDB: Introduced a compact record format that does not store the number of columns or the lengths of fixed-size columns. The old format can be requested by specifying `ROW_FORMAT=REDUNDANT`. The new format (`ROW_FORMAT=COMPACT`) is the default. The new format typically saves 20 % of disk space and memory.
- InnoDB: Setting the initial `AUTO_INCREMENT` value for an InnoDB table using `CREATE TABLE ... AUTO_INCREMENT = n` now works, and `ALTER TABLE ... AUTO_INCREMENT = n` resets the current value.
- `Seconds_Behind_Master` is `NULL` (which means “unknown”) if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. It is zero if the SQL thread has caught up to the I/O thread. It no longer grows indefinitely if the master is idle.
- The MySQL server aborts immediately instead of simply issuing a warning if it is started with the `--log-bin` option but cannot initialize the binary log at startup (that is, an error occurs when writing to the binary log file or binary log index file).
- The binary log file and binary log index file now are handled the same way as MyISAM tables when there is a “disk full” or “quota exceeded” error. See [Section A.4.3, “How MySQL Handles a Full Disk”](#).

- The MySQL server now aborts when started with the option `--log-bin-index` and without `--log-bin`, and when started with `--log-slave-updates` and without `--log-bin`.
- If the MySQL server is started without an argument to `--log-bin` and without `--log-bin-index`, thus not providing a name for the binary log index file, a warning is issued because MySQL falls back to using the hostname for that name, and this is prone to replication issues if the server's hostname's gets changed later. See [Section A.8.1, “Open Issues in MySQL”](#).
- Added account-specific `MAX_USER_CONNECTIONS` limit, which allows you to specify the maximum number of concurrent connections for the account. Also, all limited resources now are counted per account (instead of being counted per user + host pair as it was before). Use the `--old-style-user-limits` option to get the old behavior.
- InnoDB: A shared record lock (`LOCK_REC_NOT_GAP`) is now taken for a matching record in the foreign key check because inserts can be allowed into gaps.
- InnoDB: Relaxed locking in `INSERT...SELECT`, single table `UPDATE...SELECT` and single table `DELETE...SELECT` clauses when `innodb_locks_unsafe_for_binlog` is used and isolation level of the transaction is not serializable. InnoDB uses consistent read in these cases for a selected table.
- Added a new global system variable `slave_transaction_retries`: if the replication slave SQL thread fails to execute a transaction because of an InnoDB deadlock or exceeded InnoDB's `innodb_lock_wait_timeout` or NDBCluster's `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout`, it automatically retries `slave_transaction_retries` times before stopping with an error. The default is 10. (Bug #8325)
- When a client releases a user-level lock, `DO RELEASE_LOCK()` will not be written to the binary log anymore (this makes the binary log smaller); as a counterpart, the slave does not actually take the lock when it executes `GET_LOCK()`. This is mainly an optimization and should not affect existing

setups. (Bug #7998)

- The way the character set information is stored into the binary log was changed, so that it's now possible to have a replication master and slave running with different global character sets. A drawback is that replication from 5.0.3 masters to pre-5.0.3 slaves is impossible.
- The `LOAD DATA` statement was extended to support user variables in the target column list, and an optional `SET` clause. Now one can perform some transformations on data after they have been read and before they are inserted into the table. For example:

```
LOAD DATA INFILE 'file.txt'  
  INTO TABLE t1  
  (column1, @var1)  
  SET column2 = @var1/100;
```

Also, replication of `LOAD DATA` was changed, so you can't replicate such statements from a 5.0.3 master to pre-5.0.3 slaves.

- `NDB Cluster`: When using this storage engine, the output of `SHOW TABLE STATUS` now displays properly-calculated values in the `Avg_row_length` and `Data_length` columns. (Note that `BLOB` columns are not yet taken into account.) In addition, the number of replicas is now shown in the `Comment` column (as `number_of_replicas`).

Bugs fixed:

- If a `MyISAM` table on Windows had `INDEX DIRECTORY` or `DATA DIRECTORY` table options, **mysqldump** dumped the directory pathnames with single-backslash pathname separators. This would cause syntax errors when importing the dump file. **mysqldump** now changes `'\'` to `'/'` in the pathnames on Windows. (Bug #6660)
- `mysql_fix_privilege_tables` now fixes that the `mysql` privilege tables can be used in MySQL 4.1. This allows one to easily downgrade to 4.1 or run MySQL 5.0 and 4.1 with the same privilege files for testing purposes.
- Fixed bug creating user with `GRANT` fails with password but works without, (Bug #7905)

- **mysqldump** misinterpreted ‘_’ and ‘%’ characters in the names of tables to be dumped as wildcard characters. (Bug #9123)
- The definition of the enumeration-valued `sql_mode` column of the `mysql.proc` table was missing some of the current allowable SQL modes, so stored routines would not necessarily execute with the SQL mode in effect at the time of routine definition. (Bug #8902)
- `REPAIR TABLE` did not invalidate query results in the query cache that were generated from the table. (Bug #8480)
- In strict or traditional SQL mode, too-long string values assigned to string columns (`CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `TEXT`, or `BLOB`) were correctly truncated, but the server returned an `SQLSTATE` value of `01000` (should be `22001`). (Bug #6999, Bug #9029)
- Stored functions that used cursors could return incorrect results. (Bug #8386)
- `AES_DECRYPT(col_name,key)` could fail to return `NULL` for invalid values in `col_name`, if `col_name` was declared as `NOT NULL`. (Bug #8669)
- Ordering by unsigned expression (more complex than a column reference) was treating the value as signed, producing incorrectly sorted results. (Bug #7425)
- `HAVING` was treating unsigned columns as signed. (Bug #7425)
- Fixed a problem with boolean full-text searches on `utf8` columns where a double quote in the search string caused a server crash. (Bug #8351)
- For a query with both `GROUP BY` and `COUNT(DISTINCT)` clauses and a `FROM` clause with a subquery, `NULL` was returned for any `VARCHAR` column selected by the subquery. (Bug #8218)
- Fixed a bug in `TRUNCATE`, which did not work within stored procedures. A workaround has been made so that within stored procedures, `TRUNCATE` is executed like `DELETE`. This was necessary because `TRUNCATE` is implicitly locking tables. (Bug #8850)

- Fixed an optimizer bug that caused incorrectly ordered result from a query that used a FULLTEXT index to retrieve rows and there was another index that was usable for ORDER BY. For such a query, EXPLAIN showed fulltext join type, but regular (not FULLTEXT) index in the key column. (Bug #6635)
- If SELECT DISTINCT named an index column multiple times in the select list, the server tried to access different key fields for each instance of the column, which could result in a crash. (Bug #8532)
- For a stored function that refers to a given table, invoking the function while selecting from the same table resulted in a server crash. (Bug #8405)
- Comparison of a DECIMAL column containing NULL to a subquery that produced DECIMAL values resulted in a server crash. (Bug #8397)
- The --set-character-set option for **myisamchk** was changed to --set-collation. The value needed for specifying how to sort indexes is a collation name, not a character set name. (Bug #8349)
- Hostname matching didn't work if a netmask was specified for table-specific privileges. (Bug #3309)
- Corruption of MyISAM table indexes could occur with TRUNCATE TABLE if the table had already been opened. For example, this was possible if the table had been opened implicitly by selecting from a MERGE table that mapped to the MyISAM table. The server now issues an error message for TRUNCATE TABLE under these conditions. (Bug #8306)
- Setting the connection collation to a value different from the server collation followed by a CREATE TABLE statement that included a quoted default value resulted in a server crash. (Bug #8235)
- Fixed handling of table-name matching in **mysqlhotcopy** to accommodate DBD::mysql 2.9003 and up (which implement identifier quoting). (Bug #8136)
- Selecting from a view defined as a join caused a server crash if the query cache was enabled. (Bug #8054)
- Results in the query cache generated from a view were not properly

invalidated after ALTER VIEW or DROP VIEW on that view. (Bug #8050)

- FOUND_ROWS() returned an incorrect value after a SELECT SQL_CALC_FOUND_ROWS DISTINCT statement that selected constants and included GROUP BY and LIMIT clauses. (Bug #7945)
- Selecting from an INFORMATION_SCHEMA table combined with a subquery on an INFORMATION_SCHEMA table caused an error with the message Table tbl_name is corrupted. (Bug #8164)
- Fixed a problem with equality propagation optimization for prepared statements and stored procedures that caused a server crash upon re-execution of the prepared statement or stored procedure. (Bug #8115, Bug #8849)
- LEFT OUTER JOIN between an empty base table and a view on an empty base table caused a server crash. (Bug #7433)
- Use of GROUP_CONCAT() in the select list when selecting from a view caused a server crash. (Bug #7116)
- Use of a view in a correlated subquery that contains HAVING but no GROUP BY caused a server crash. (Bug #6894)
- Handling by mysql_list_fields() of references to stored functions within views was incorrect and could result in a server crash. (Bug #6814)
- **mysqldump** now avoids writing SET NAMES to the dump output if the server is older than version 4.1 and would not understand that statement. (Bug #7997)
- Fixed problems when selecting from a view that had an EXISTS or NOT EXISTS subquery. Selecting columns by name caused a server crash. With SELECT *, a crash did not occur, but columns in outer query were not resolved properly. (Bug #6394)
- DDL statements for views were not being written to the binary log (and thus not subject to replication). (Bug #4838)
- The CHAR() function was not ignoring NULL arguments, contrary to the

documentation. (Bug #6317)

- Creating a table using a name containing a character that is illegal in `character_set_client` resulted in the character being stripped from the name and no error. The character now is considered an error. (Bug #8041)
- Fixed a problem with the Cyrillic letters I and SHORT I being treated the same by the `utf8_general_ci` collation. (Bug #8385)
- Some `INFORMATION_SCHEMA` columns that contained catalog identifiers were of type `LONGTEXT`. These were changed to `VARCHAR(N)`, where *N* is the appropriate maximum identifier length. (Bug #7215)
- Some `INFORMATION_SCHEMA` columns that contained timestamp values were of type `VARBINARY`. These were changed to `TIMESTAMP`. (Bug #7217)
- An expression that tested a case-insensitive character column against string constants that differed in lettercase could fail because the constants were treated as having a binary collation. (For example, `WHERE city='London' AND city='london'` could fail.) (Bug #7098, Bug #8690)
- The output of the `STATUS (\s)` command in **mysql** had the values for the server and client character sets reversed. (Bug #7571)
- If the slave was running with `--replicate-*-table` options which excluded one temporary table and included another, and the two tables were used in a single `DROP TEMPORARY TABLE IF EXISTS` statement, as the ones the master automatically writes to its binary log upon client's disconnection when client has not explicitly dropped these, the slave could forget to delete the included replicated temporary table. Only the slave needs to be upgraded. (Bug #8055)
- When setting integer system variables to a negative value with `SET VARIABLES`, the value was treated as a positive value modulo 2^{32} . (Bug #6958)
- Corrected a problem with references to `DUAL` where statements such as `SELECT 1 AS a FROM DUAL` would succeed but statements such as `SELECT 1 AS a FROM DUAL LIMIT 1` would fail. (Bug #8023)

- Fixed a server crash caused by `DELETE FROM tbl_name ... WHERE ... ORDER BY tbl_name.col_name` when the `ORDER BY` column was qualified with the table name. (Bug #8392)
- Fixed a bug in `MATCH ... AGAINST` in natural language mode that could cause a server crash if the `FULLTEXT` index was not used in a join (`EXPLAIN` did not show `fulltext` join mode) and the search query matched no rows in the table (Bug #8522).
- InnoDB: Honor the `--tmpdir` startup option when creating temporary files. Previously, InnoDB temporary files were always created in the temporary directory of the operating system. On Netware, InnoDB will continue to ignore `--tmpdir`. (Bug #5822)
- Platform and architecture information in version information produced for `--version` option on Windows was always `win95/win98 (i32)`. More accurately determine platform as `win32` or `win64` for 32-bit or 64-bit Windows, and architecture as `ia32` for x86, `ia64` for Itanium, and `axp` for Alpha. (Bug #4445)
- If multiple semicolon-separated statements were received in a single packet, they were written to the binary log as a single event rather than as separate per-statement events. For a server serving as a replication master, this caused replication to fail when the event was sent to slave servers. (Bug #8436)
- Fixed `LOAD INDEX` statement to actually load index in memory. (Bug #8452)
- Fixed a failure of multiple-table updates to replicate properly on slave servers when `--replicate-*-table` options had been specified. (Bug #7011)
- Fixed failure of `CREATE TABLE ... LIKE` Windows when the source or destination table was located in a symlinked database directory. (Bug #6607)
- With `lower_case_table_names` set to 1, **mysqldump** on Windows could write the same table name in different lettercase for different SQL statements. Fixed so that consistent lettercase is used. (Bug #5185)

- **mysqld_safe** now understands the `--help` option. Previously, it ignored the option and attempted to start the server anyway. (Bug #7931)
- Fixed problem in `NO_BACKSLASH_ESCAPES` SQL mode for strings that contained both the string quoting character and backslash. (Bug #6368)
- Fixed some portability issues with overflow in floating point values.
- Prepared statements now gives warnings on prepare.
- Fixed bug in prepared statements with `SUM(DISTINCT . . .)`.
- Fixed bug in prepared statements with `OUTER JOIN`.
- Fixed a bug in `CONV()` function returning unsigned `BIGINT` number (third argument is positive, and return value does not fit in 32 bits). (Bug #7751)
- Fixed a failure of the `IN()` operator to return correct result if all values in the list were constants and some of them were using substring functions, for example, `LEFT()`, `RIGHT()`, or `MID()`. (Bug #7716)
- Fixed a crash in `CONVERT_TZ()` function when its second or third argument was from a const table (see [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#)). (Bug #7705)
- Fixed a problem with calculation of number of columns in row comparison against subquery. (Bug #8020)
- Fixed erroneous output resulting from `SELECT DISTINCT` combined with a subquery and `GROUP BY`. (Bug #7946)
- Fixed server crash in comparing a nested row expression (for example `row(1, (2, 3))`) with a subquery. (Bug #8022)
- Fixed server crash resulting from certain correlated subqueries with forward references (references to an alias defined later in the outer query). (Bug #8025)
- Fixed server crash resulting from re-execution of prepared statements containing subqueries. (Bug #8125)

- Fixed a bug where ALTER TABLE improperly would accept an index on a TIMESTAMP column that CREATE TABLE would reject. (Bug #7884)
- SHOW CREATE TABLE now reports ENGINE=MEMORY rather than ENGINE=HEAP for a MEMORY table (unless the MYSQL323 SQL mode is enabled). (Bug #6659)
- Fixed a bug where the use of GROUP_CONCAT() with HAVING caused a server crash. (Bug #7769)
- Fixed a bug where comparing the result of a subquery to a non-existent column caused a server crash on Windows. (Bug #7885)
- Fixed a bug in a combination of -not and trunc* operators of full-text search. Using more than one truncated negative search term, was causing empty result set.
- InnoDB: Corrected the handling of trailing spaces in the ucs2 character set. (Bug #7350, Bug #8771)
- InnoDB: Use native tmpfile() function on Netware. All InnoDB temporary files are created under sys:\tmp. Previously, InnoDB temporary files were never deleted on Netware.
- Fixed a bug in max_heap_table_size handling, that resulted in Table is full error when the table was still smaller than the limit. (Bug #7791).
- Fixed a symlink vulnerability in the **mysqlaccess** script. Reported by Javier Fernandez-Sanguino Pena and [Debian Security Audit Team](#). ([CVE-2005-0004](#))
- Fixed a bug that caused server crash if some error occurred during filling of temporary table created for derived table or view handling. (Bug #7413)
- Fixed a bug which caused server crash if query containing CONVERT_TZ() function with constant arguments was prepared. (Bug #6849)
- Prevent adding CREATE TABLE ... SELECT query to the binary log when the insertion of new records partially failed. (Bug #6682)

- Fixed a bug which caused a crash when only the slave I/O thread was stopped and started. (Bug #6148)
- Giving **mysqld** a SIGHUP caused it to crash.
- Changed semantics of CREATE/ALTER/DROP DATABASE statements so that replication of CREATE DATABASE is possible when using --binlog-do-db and --binlog-ignore-db. (Bug #6391)
- A sequence of BEGIN (or SET AUTOCOMMIT=0), FLUSH TABLES WITH READ LOCK, transactional update, COMMIT, FLUSH TABLES WITH READ LOCK could hang the connection forever and possibly the MySQL server itself. This happened for example when running the innobackup script several times. (Bug #6732)
- **mysqlbinlog** did not print SET PSEUDO_THREAD_ID statements in front of LOAD DATA INFILE statements inserting into temporary tables, thus causing potential problems when rolling forward these statements after restoring a backup. (Bug #6671)
- InnoDB: Fixed a bug no error message for ALTER with InnoDB and AUTO_INCREMENT (Bug #7061). InnoDB now supports ALTER TABLE...AUTO_INCREMENT = x query to set auto increment value for a table.
- Made the MySQL server accept executing SHOW CREATE DATABASE even if the connection has an open transaction or locked tables; refusing it made **mysqldump --single-transaction** sometimes fail to print a complete CREATE DATABASE statement for some dumped databases. (Bug #7358)
- Fixed that, when encountering a “disk full” or “quota exceeded” write error, MyISAM sometimes didn't sleep and retry the write, thus resulting in a corrupted table. (Bug #7714)
- Fixed that --expire-log-days was not honored if using only transactions. (Bug #7236)
- Fixed that a slave could crash after replicating many ANALYZE TABLE, OPTIMIZE TABLE, or REPAIR TABLE statements from the master. (Bug #6461, Bug #7658)

- **mysqlbinlog** forgot to add backquotes around the collation of user variables (causing later parsing problems as BINARY is a reserved word). (Bug #7793)
- Ensured that **mysqldump --single-transaction** sets its transaction isolation level to REPEATABLE READ before proceeding (otherwise if the MySQL server was configured to run with a default isolation level lower than REPEATABLE READ it could give an inconsistent dump). (Bug #7850)
- Fixed that when using the RPAD() function (or any function adding spaces to the right) in a query that had to be resolved by using a temporary table, all resulting strings had rightmost spaces removed (that is, RPAD() did not work) (Bug #4048)
- Fixed that a 5.0.3 slave can connect to a master < 3.23.50 without hanging (the reason for the hang is a bug in these quite old masters -- SELECT @@unknown_var hangs them -- which was fixed in MySQL 3.23.50). (Bug #7965)
- InnoDB: Fixed a deadlock without any locking, simple select and update (Bug #7975). InnoDB now takes an exclusive lock when INSERT ON DUPLICATE KEY UPDATE is checking duplicate keys.
- Fixed a bug where MySQL was allowing concurrent updates (inserts, deletes) to a table if binary logging is enabled. Changed to ensure that all updates are executed in a serialized fashion, because they are executed serialized when binlog is replayed. (Bug #7879)
- Fixed a rare race condition which could lead to FLUSH TABLES WITH READ LOCK hanging. (Bug #8682)
- Fixed a bug in replication that caused the master to stamp generated statements (such as SET commands) with an error_code intended only for another statement. This could happen, for example, when a statements generates a duplicate key error on the master but must be replicated. (Bug #8412)

D.1.25. Changes in release 5.0.2 (01 December 2004)

Functionality added or changed:

- **Warning: Incompatible change!** The precedence of NOT operator has changed so that expressions such as NOT a BETWEEN b AND c are parsed correctly as NOT (a BETWEEN b AND c) rather than as (NOT a) BETWEEN b AND c. The pre-5.0 higher-precedence behavior can be obtained by enabling the new HIGH_NOT_PRECEDENCE SQL mode.
- **Warning: Incompatible change!** SHOW STATUS now shows the session (thread-specific) status variables and SHOW GLOBAL STATUS shows the status variables for the whole server.

Before MySQL 5.0.2, SHOW STATUS returned global status values. Because the default as of 5.0.2 is to return session values, this is incompatible with previous versions. To issue a SHOW STATUS statement that will retrieve global status values for all versions of MySQL, write it like this:

```
SHOW /*!50002 GLOBAL */ STATUS;
```

- Added support for the INFORMATION_SCHEMA “information database” that provides database metadata. See [Chapter 20, The INFORMATION_SCHEMA Database](#).
- A HAVING clause in a SELECT statement now can refer to columns in the GROUP BY clause, as required by standard SQL.
- Added the CREATE USER and RENAME USER statements.
- Modify DROP USER so that it drops the account, including all its privileges. Formerly, it removed the account record only for an account that had had all privileges revoked.
- Added IS [NOT] boolean_value syntax, where *boolean_value* is TRUE, FALSE, or UNKNOWN.
- Added several InnoDB status variables. See [Section 5.2.4, “Server Status Variables”](#).
- Implemented the WITH CHECK OPTION clause for CREATE VIEW.
- CHECK TABLE now works for views.

- The `SCHEMA` and `SCHEMAS` keywords are now accepted as synonyms for `DATABASE` and `DATABASES`.
- Added initial support for rudimentary triggers (the `CREATE TRIGGER` and `DROP TRIGGER` statements).
- Added basic support for read-only server side cursors.
- **`mysqldump --single-transaction --master-data`** is now able to take an online (non-blocking) dump of InnoDB and report the corresponding binary log coordinates, which makes a backup suitable for point-in-time recovery, roll-forward or replication slave creation. See [Section 8.12, “mysqldump — A Database Backup Program”](#).
- Added `--start-datetime`, `--stop-datetime`, `--start-position`, `--stop-position` options to **`mysqlbinlog`** (makes point-in-time recovery easier).
- Made the MySQL server not react to signals `SIGHUP` and `SIGQUIT` on Mac OS X 10.3. This is needed because under this OS, the MySQL server receives lots of these signals (reported as Bug #2030).
- New `--auto-increment-increment` and `--auto-increment-offset` startup options. These allow you to set up a server to generate auto-increment values that don't conflict with another server.
- MySQL now by default checks dates and in strict mode allows only fully correct dates. If you want MySQL to behave as before, you should enable the new `ALLOW_INVALID_DATES` SQL mode.
- Added `STRICT_TRANS_TABLES`, `STRICT_ALL_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`, `ERROR_FOR_DIVISION_BY_ZERO`, and `TRADITIONAL` SQL modes. The `TRADITIONAL` mode is shorthand for all the preceding modes. When using mode `TRADITIONAL`, MySQL generates an error if you try to insert a wrong value in a column. It does not adjust the value to the closest possible legal value.
- MySQL now remembers which columns were declared to have default values. In `STRICT_TRANS_TABLES/STRICT_ALL_TABLES` mode, you now get an error if you do an `INSERT` without specifying all columns that don't have a default value. A side effect of this is that when you do `SHOW CREATE` for a

new table, you no longer see a `DEFAULT` value for a column for which you didn't specify a default value.

- The compilation flag `DONT_USE_DEFAULT_FIELDS` was removed because you can get the same behavior by setting the `sql_mode` system variable to `STRICT_TRANS_TABLES`.
- Added `NO_AUTO_CREATE_USER` SQL mode to prevent `GRANT` from automatically creating new users if it would otherwise do so, unless a password also is specified.
- We now detect too-large floating point numbers during statement parsing and generate an error messages for them.
- Renamed the `sql_updatable_view_key` system variable to `updatable_views_with_limit`. This variable now can have only two values:
 - `1` or `YES`: Don't issue an error message (warning only) if a `VIEW` without presence of a key in the underlying table is used in queries with a `LIMIT` clause for updating. (This is the default value.)
 - `0` or `NO`: Prohibit update of a `VIEW`, which does not contain a key in the underlying table and the query uses a `LIMIT` clause (usually get from GUI tools).
- Reverted output format of `SHOW TABLES` to old pre-5.0.1 format that did not include a table type column. To get the additional column that lists the table type, use `SHOW FULL TABLES NOW`.
- The **`mysql_fix_privilege_tables`** script now initializes the global `CREATE VIEW` and `SHOW VIEW` privileges in the user table to the value of the `CREATE` privilege in that table.
- If the server finds that the user table has not been upgraded to include the view-related privilege columns, it treats each account as having view privileges that are the same as its `CREATE` privilege.
- InnoDB: If you specify the option `innodb_locks_unsafe_for_binlog` in `my.cnf`, InnoDB in an `UPDATE` or a `DELETE` only locks the rows that it

updates or deletes. This greatly reduces the probability of deadlocks.

- A connection doing a rollback now displays "Rolling back" in the State column of SHOW PROCESSLIST.
- **mysqlbinlog** now prints an informative commented line (thread id, timestamp, server id, and so forth) before each LOAD DATA INFILE, like it does for other queries; unless --short-form is used.
- Two new server system variables were introduced. auto_increment_increment and auto_increment_offset can be set locally or globally, and are intended for use in controlling the behavior of AUTO_INCREMENT columns in master-to-master replication. Note that these variables are not intended to take the place of sequences. See [Section 5.2.2, "Server System Variables"](#).

Bugs fixed:

- Fixed that **mysqlbinlog --read-from-remote-server** sometimes couldn't accept two binary log files on the command line. (Bug #4507)
- Fixed that **mysqlbinlog --position --read-from-remote-server** had incorrect # at lines. (Bug #4506)
- Fixed that CREATE TABLE ... TYPE=HEAP ... AS SELECT... caused replication slave to stop. (Bug #4971)
- Fixed that mysql_options(..., MYSQL_OPT_LOCAL_INFILE, ...) failed to disable LOAD DATA LOCAL INFILE. (Bug #5038)
- Fixed that disable-local-infile option had no effect if client read it from a configuration file using mysql_options(..., MYSQL_READ_DEFAULT, ...). (Bug #5073)
- Fixed that SET GLOBAL SYNC_BINLOG did not work on some platforms (Mac OS X). (Bug #5064)
- Fixed that **mysql-test-run** failed on the rpl_trunc_binlog test if running test from the installed (the target of 'make install') directory. (Bug #5050)

- Fixed that **mysql-test-run** failed on the `grant_cache` test when run as Unix user 'root'. (Bug #4678)
- Fixed an unlikely deadlock which could happen when using `KILL`. (Bug #4810)
- Fixed a crash when one connection got `KILLED` while it was doing `START SLAVE`. (Bug #4827)
- Made `FLUSH TABLES WITH READ LOCK` block `COMMIT` if server is running with binary logging; this ensures that the binary log position can be trusted when doing a full backup of tables and the binary log. (Bug #4953)
- Fixed that the counter of an `auto_increment` column was not reset by `TRUNCATE TABLE` if the table was a temporary one. (Bug #5033)
- Fixed slave SQL thread so that the `SET COLLATION_SERVER...` statements it replicates don't advance its position (so that if it gets interrupted before the actual update query, it later redoes the `SET`). (Bug #5705)
- Fixed that if the slave SQL thread found a syntax error in a query (which should be rare, as the master parsed it successfully), it stops. (Bug #5711)
- Fixed that if a write to a MyISAM table fails because of a full disk or an exceeded disk quota, it prints a message to the error log every 10 minutes, and waits until disk becomes free. (Bug #3248)
- Fixed problem introduced in 4.0.21 where a connection starting a transaction, doing updates, then `FLUSH TABLES WITH READ LOCK`, then `COMMIT`, would cause replication slaves to stop (complaining about error 1223). Bug surfaced when using the InnoDB `innobackup` script. (Bug #5949)
- `OPTIMIZE TABLE`, `REPAIR TABLE`, and `ANALYZE TABLE` are now replicated without any error code in the binary log. (Bug #5551)
- If a connection had an open transaction but had done no updates to transactional tables (for example if had just done a `SELECT FOR UPDATE` then executed a non-transactional update, that update automatically committed the transaction (thus releasing InnoDB's row-level locks etc).

(Bug #5714)

- If a connection was interrupted by a network error and did a rollback, the network error code got stored into the BEGIN and ROLLBACK binary log events; that caused superfluous slave stops. (Bug #6522)
- Fixed a bug which prevented **mysqlbinlog** from being able to read from stdin, for example, when piping the output from **zcat** to **mysqlbinlog**. (Bug #7853)

D.1.26. Changes in release 5.0.1 (27 July 2004)

Note: This build passes our test suite and fixes a lot of reported bugs found in the previous 5.0.0 release. However, please be aware that this is not a “standard MySQL build” in the sense that there are still some open critical bugs in our bugs database at <http://bugs.mysql.com/> that affect this release as well. We are actively fixing these and will make a new release where these are fixed as soon as possible. However, this binary should be a good candidate for testing new MySQL 5.0 features for future products.

Functionality added or changed:

- **Warning: Incompatible change!** C API change: `mysql_shutdown()` now requires a second argument. This is a source-level incompatibility that affects how you compile client programs; it does not affect the ability of compiled clients to communicate with older servers. See [Section 22.2.3.64, “mysql_shutdown\(\)”](#).
- When installing a MySQL server as a Windows service, the installation command can include a `--local-service` option following the service name to cause the server to run using the `LocalService` Windows account that has limited privileges. This is in addition to the `--defaults-file` option that also can be given following the service name.
- Added support for read-only and updatable views based on a single table or other updatable views. View use requires that you upgrade your grant tables to add the view-related privileges. See [Section 5.6.1, “mysql_fix_privilege_tables — Upgrade MySQL System Tables”](#).

- Implemented a new “greedy search” optimizer that can significantly reduce the time spent on query optimization for some many-table joins. (You are affected if not only some particular SELECT is slow, but even using EXPLAIN for it takes a noticeable amount of time.) Two new system variables, optimizer_search_depth and optimizer_prune_level, can be used to fine-tune optimizer behavior.
- A stored procedure is no longer “global.” That is, it now belongs to a specific database:
 - When a database is dropped, all routines belonging to that database are also dropped.
 - Procedure names may be qualified, for example, db.p()
 - When executed from another database, an implicit USE db_name is in effect.
 - Explicit USE db_name statements no longer are allowed in a stored procedure.

See [Chapter 17, Stored Procedures and Functions](#).

- Fixed SHOW TABLES output field name and values according to standard. Field name changed from Type to table_type, values are BASE TABLE, VIEW and ERROR. (Bug #4603)
- Added the sql_updatable_view_key system variable.
- Added the --replicate-same-server-id server option.
- Added Last_query_cost status variable that reports optimizer cost for last compiled query.
- Added the --to-last-log option to **mysqlbinlog**, for use in conjunction with --read-from-remote-server.
- Added the --innodb-safe-binlog server option, which adds consistency guarantees between the content of InnoDB tables and the binary log. See [Section 5.12.3, “The Binary Log”](#).

- `OPTIMIZE TABLE` for InnoDB tables is now mapped to `ALTER TABLE` instead of `ANALYZE TABLE`. This rebuilds the table, which updates index statistics and frees space in the clustered index.
- `sync_frm` is now a settable global variable (not only a startup option).
- For replication of MEMORY (HEAP) tables: Made the master automatically write a `DELETE FROM` statement to its binary log when a MEMORY table is opened for the first time since master's startup. This is for the case where the slave has replicated a non-empty MEMORY table, then the master is shut down and restarted: the table is now empty on master; the `DELETE FROM` empties it on slave too. Note that even with this fix, between the master's restart and the first use of the table on master, the slave still has out-of-date data in the table. But if you use the `--init-file` option to populate the MEMORY table on the master at startup, it ensures that the failing time interval is zero. (Bug #2477)
- When a session having open temporary tables terminates, the statement automatically written to the binary log is now `DROP TEMPORARY TABLE IF EXISTS` instead of `DROP TEMPORARY TABLE`, for more robustness.
- The MySQL server now returns an error if `SET SQL_LOG_BIN` is issued by a user without the SUPER privilege (in previous versions it just silently ignored the statement in this case).
- Changed that when the MySQL server has binary logging disabled (that is, no `--log-bin` option was used), then no transaction binary log cache is allocated for connections. This should save `binlog_cache_size` bytes of memory (32KB by default) for every connection.
- Added the `sync_binlog=N` global variable and startup option, which makes the MySQL server synchronize its binary log to disk (`fdatasync()`) after every Nth write to the binary log.
- Changed the slave SQL thread to print less useless error messages (no more message duplication; no more messages when an error is skipped because of `slave-skip-errors`).
- `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, single-table `DELETE`, and single-table `UPDATE` now are written to the binary log even if they

changed nothing on the master (for example, even if a DELETE matched no rows). The old behavior sometimes caused bad surprises in replication setups.

- Replication and **mysqlbinlog** now have better support for the case that the session character set and collation variables are changed within a given session. See [Section 6.7, “Replication Features and Known Problems”](#).
- Killing a CHECK TABLE statement does not result in the table being marked as “corrupted” any more; the table remains as if CHECK TABLE had not even started. See [Section 13.5.5.3, “KILL Syntax”](#).

Bugs fixed:

- Strange results with index (x, y) ... WHERE x=val_1 AND y>=val_2 ORDER BY pk; (Bug #3155)
- Adding ORDER BY to a query that uses a subquery can cause incorrect results. (Bug #3118)
- ALTER DATABASE caused the client to hang if the database did not exist. (Bug #2333)
- SLAVE START (which is a deprecated syntax, START SLAVE should be used instead) could crash the slave. (Bug #2516)
- Multiple-table DELETE statements were never replicated by the slave if there were any --replicate-* -table options. (Bug #2527)
- The MySQL server did not report any error if a statement (submitted through mysql_real_query() or mysql_stmt_prepare()) was terminated by garbage characters. This can happen if you pass a wrong length parameter to these functions. The result was that the garbage characters were written into the binary log. (Bug #2703)
- Replication: If a client connects to a slave server and issues an administrative statement for a table (for example, OPTIMIZE TABLE or REPAIR TABLE), this could sometimes stop the slave SQL thread. This does not lead to any corruption, but you must use START SLAVE to get replication going again. (Bug #1858)

- Made clearer the error message that one gets when an update is refused because of the `--read-only` option. (Bug #2757)
- Fixed that `--replicate-wild-*` table rules apply to `ALTER DATABASE` when the table pattern is `%`, as is the case for `CREATE DATABASE` and `DROP DATABASE`. (Bug #3000)
- Fixed that when a Rotate event is found by the slave SQL thread in the middle of a transaction, the value of `Relay_Log_Pos` in `SHOW SLAVE STATUS` remains correct. (Bug #3017)
- Corrected the master's binary log position that InnoDB reports when it is doing a crash recovery on a slave server. (Bug #3015)
- Changed the column `Seconds_Behind_Master` in `SHOW SLAVE STATUS` to never show a value of `-1`. (Bug #2826)
- Changed that when a `DROP TEMPORARY TABLE` statement is automatically written to the binary log when a session ends, the statement is recorded with an error code of value zero (this ensures that killing a `SELECT` on the master does not result in a superfluous error on the slave). (Bug #3063)
- Changed that when a thread handling `INSERT DELAYED` (also known as a `delayed_insert` thread) is killed, its statements are recorded with an error code of value zero (killing such a thread does not endanger replication, so we thus avoid a superfluous error on the slave). (Bug #3081)
- Fixed deadlock when two `START SLAVE` commands were run at the same time. (Bug #2921)
- Fixed that a statement never triggers a superfluous error on the slave, if it must be excluded given the `--replicate-*` options. The bug was that if the statement had been killed on the master, the slave would stop. (Bug #2983)
- The `--local-load` option of **mysqlbinlog** now requires an argument.
- Fixed a segmentation fault when running `LOAD DATA FROM MASTER` after `RESET SLAVE`. (Bug #2922)
- **mysqlbinlog --read-from-remote-server** read all binary logs following the

one that was requested. It now stops at the end of the requested file, the same as it does when reading a local binary log. There is an option `--to-last-log` to get the old behavior. (Bug #3204)

- Fixed **mysqlbinlog --read-from-remote-server** to print the exact positions of events in the "at #" lines. (Bug #3214)
- Fixed a rare error condition that caused the slave SQL thread spuriously to print the message Binlog has bad magic number and stop when it was not necessary to do so. (Bug #3401)
- Fixed **mysqlbinlog** not to forget to print a USE statement under rare circumstances where the binary log contained a LOAD DATA INFILE statement. (Bug #3415)
- Fixed a memory corruption when replicating a LOAD DATA INFILE when the master had version 3.23. (Bug #3422)
- Multiple-table DELETE statements were always replicated by the slave if there were some `--replicate-*--ignore-table` options and no `--replicate-*--do-table` options. (Bug #3461)
- Fixed a crash of the MySQL slave server when it was built with `--with-debug` and replicating itself. (Bug #3568)
- Fixed that in some replication error messages, a very long query caused the rest of the message to be invisible (truncated), by putting the query last in the message. (Bug #3357)
- If `server-id` was not set using startup options but with SET GLOBAL, the replication slave still complained that it was not set. (Bug #3829)
- **mysql_fix_privilege_tables** didn't correctly handle the argument of its `--password=password_val` option. (Bug #4240)
- Fixed potential memory overrun in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017, [CVE-2004-0836](#))
- During the installation process of the server RPM on Linux, **mysqld** was

run as the root system user, and if you had `--log-bin=somewhere_out_of_var_lib_mysql` it created binary log files owned by root in this directory, which remained owned by root after the installation. This is now fixed by starting **mysqld** as the mysql system user instead. (Bug #4038)

- Made `DROP DATABASE` honor the value of `lower_case_table_names`. (Bug #4066)
- The slave SQL thread refused to replicate `INSERT ... SELECT` if it examined more than 4 billion rows. (Bug #3871)
- **mysqlbinlog** didn't escape the string content of user variables, and did not deal well when these variables were in non-ASCII character sets; this is now fixed by always printing the string content of user variables in hexadecimal. The character set and collation of the string is now also printed. (Bug #3875)
- Fixed incorrect destruction of expression that led to a server crash on complex `AND/OR` expressions if query was ignored (either by a replication server because of `--replicate-*-table` rules, or by any MySQL server because of a syntax error). (Bug #3969, Bug #4494)
- If `CREATE TEMPORARY TABLE t SELECT` failed while loading the data, the temporary table was not dropped. (Bug #4551)
- Fixed that when a multiple-table `DROP TABLE` failed to drop a table on the master server, the error code was not written to the binary log. (Bug #4553)
- When the slave SQL thread was replicating a `LOAD DATA INFILE` statement, it didn't show the statement in the output of `SHOW PROCESSLIST`. (Bug #4326)

D.1.27. Changes in release 5.0.0 (22 December 2003: Alpha)

Functionality added or changed:

- The output of the `SHOW BINLOG EVENTS` statement has been modified. The `orig_log_pos` column has been renamed to `End_log_pos` and now represents the offset of the last byte of the event, plus one.

- **Important note:** If you upgrade to MySQL 4.1.1 or higher, it is difficult to downgrade back to 4.0 or 4.1.0! That is because, for earlier versions, InnoDB is not aware of multiple tablespaces.
- Added support for SUM(DISTINCT), MIN(DISTINCT), and MAX(DISTINCT).
- The KILL statement now takes CONNECTION and QUERY modifiers. The first is the same as KILL with no modifier (it kills a given connection thread). The second kills only the statement currently being executed by the connection.
- Added TIMESTAMPADD() and TIMESTAMPDIFF() functions.
- Added WEEK and QUARTER values as INTERVAL arguments for the DATE_ADD() and DATE_SUB() functions.
- New binary log format that enables replication of these session variables: sql_mode, SQL_AUTO_IS_NULL, FOREIGN_KEY_CHECKS (which was replicated since 4.0.14, but here it's done more efficiently and takes less space in the binary logs), UNIQUE_CHECKS. Other variables (like character sets, SQL_SELECT_LIMIT, ...) will be replicated in upcoming 5.0.x releases.
- Implemented Index Merge optimization for OR clauses. See [Section 7.2.6, “Index Merge Optimization”](#).
- Basic support for stored procedures (SQL:2003 style). See [Chapter 17, *Stored Procedures and Functions*](#).
- Added SELECT INTO list_of_vars, which can be of mixed (that is, global and local) types. See [Section 17.2.7.3, “SELECT ... INTO Statement”](#).
- Easier replication upgrade (5.0.0 masters can read older binary logs and 5.0.0 slaves can read older relay logs). See [Section 6.5, “Replication Compatibility Between MySQL Versions”](#), for more details). The format of the binary log and relay log is changed compared to that of MySQL 4.1 and older.

Bugs fixed:

D.2. Changes in MySQL Cluster

Starting from 4.1.13 and 5.0.7, all Cluster changes are included in the MySQL Change History, and this manual section is no longer separately maintained.

D.2.1. Changes in MySQL Cluster-5.0.7 (10 June 2005)

Note: Starting with version 5.0.8, changes for MySQL Cluster can be found in the combined MySQL Change History.

Functionality added or changed:

Bugs fixed:

- (Bug #11019) mgmapi start backup in some cases returns wrong backupid
- (Bug #10190) Backup from cluster with NoOfReplica=1 is corrupt
- (Bug #9246) Condition pushdown and left join, wrong result
- (Bug #10956) More than 7 node restarts with `--initial` caused cluster to fail.
- (Bug #9945) `ALTER TABLE` caused server crash. (Linux/390)
- (Bug #9826) (Bug #10948) Schema change (`DROP TABLE`, `ALTER TABLE`) crashed HPUX and PPC32.
- (Bug #10711) (Bug #9363) (Bug #8918) (Bug #10058) (Bug #9025) Cluster would time out and crash after first query; setting `DataMemory` to more than 2GB prevented cluster from starting; calling `ndb_select_count()` crashed the cluster. (64-bit Unix OSes)

D.2.2. Changes in MySQL Cluster-5.0.6 (26 May 2005)

Functionality added or changed:

- Limit on number of metadata objects (number of tables, indexes and BLOBs) now increased to 20,320

Bugs fixed:

- The server would hang on successive calls to an `INSERT ... ON DUPLICATE KEY UPDATE` query. (Bug #9725)
- (Bug #10193) Invalid DataDir in config causes ndbd segmentation fault
- (Bug #10813) Build with SCI Transporter fails
- (Bug #10831) ndb mgmd LogDestination maxfiles does not rotate logs properly

D.2.3. Changes in MySQL Cluster-5.0.5 (Not released)

Functionality added or changed:

- Decreased IndexMemory Usage
- Parallel key lookup (read-multi-range) for queries like `SELECT * FROM t1 WHERE primary_key IN (1,2,3,4,5,6,7,8,9,10);`

Bugs fixed:

Patches merged from versions 4.1.11 and 4.1.12

- (Bug #8315) NdbScanFilter cmp method only works for strings of exact word boundary length
- (Bug #8103) Configuration handling error
- (Bug #8035) mysqld signal 10 when ndbd is shutdown
- (Bug #7631) NDB\$EVENT contains unreadable event and table names
- (Bug #7628) Filtered event types are ignored
- (Bug #7627) Drop Event operation fails

- (Bug #7424) create index on datetime fails

D.2.4. Changes in MySQL Cluster-5.0.4 (16 April 2005)

Functionality added or changed:

- Condition pushdown to storage engine now works for update and delete as well

Bugs fixed:

- (Bug #9675) Auto-increment not working with INSERT..SELECT and NDB storage
- (Bug #9517) Condition pushdown to storage engine does not work for update/delete
- (Bug #9282) API Node Crashes/Reloads on 'DELETE FROM'
- (Bug #9280) Memory leak in cluster when dependent sub-queries are used
- (Bug #8585) ndb_cache2 fails on aix52

D.2.5. Changes in MySQL Cluster-5.0.3 (23 March 2005: Beta)

Functionality added or changed:

- Condition pushdown to storage engine
- Query cache enabled for cluster

Bugs fixed:

- Patches merged from version 4.1.10

D.2.6. Changes in MySQL Cluster-5.0.1 (27 July 2004)

Functionality added or changed:

- This was the first MySQL Cluster release in the 5.0 series. As nearly all

attention was still focused on getting 4.1 stable, it is not recommended to use MySQL 5.0.1 for MySQL Cluster.

Bugs fixed:

- N/A

D.2.7. Changes in MySQL Cluster-4.1.13 (15 July 2005)

Functionality added or changed:

Bugs fixed:

- (Bug #11132) Connections between data nodes and management nodes were not being closed following shutdown of `ndb_mgmd`.
- (Bug #11050) `ndb_mgm> show` printed incorrectly after master data node failure.
- (Bug #10956) More than 7 node restarts with `--initial` caused cluster to fail.
- (Bug #9826) (Bug #10948) Schema change (`DROP TABLE`, `ALTER TABLE`) crashed HPUX and PPC32.
- (Bug #9025) Data nodes failed to restart on 64-bit Solaris.
- (Bug #11166) Insert records were incorrectly applied by `ndb_restore`, thus making restoration from backup inconsistent if the binlog contained inserts.
- (Bug #8918) (Bug #9363) (Bug #10711) (Bug #10058) (Bug #9025) Cluster would time out and crash after first query; setting `DataMemory` to more than 2GB prevented cluster from starting; calling `ndb_select_count()` crashed the cluster. (64-bit Unix OSes)
- (Bug #10190) When making a backup of a cluster where `NumberOfReplicas` was equal to 1, the backup's metadata was corrupted. (Linux)
- (Bug #9945) `ALTER TABLE` caused server crash. (Linux/390)

- (Bug #11133) A delete operation performed as part of a transaction caused an erroneous result.
- (Bug #10294) Not allowing sufficient parallelism in cluster configuration (for example, NoOfTransactions too small) caused `ndb_restore` to fail without generating any error messages.
- (Bug #11290) Setting `TransactionInactiveTimeout= 0` did not result in an infinite timeout.

D.2.8. Changes in MySQL Cluster-4.1.12 (13 May 2005)

Functionality added or changed:

Bugs fixed:

- (Bug #10471) Backup can become inconsistent with certain combinations of multiple-row updates
- (Bug #10287) `ndb_select_all "delimiter"` option non functional
- (Bug #10142) Unhandled resource shortage in UNIQUE index code
- (Bug #10029) crash in ordered index scan after db full
- (Bug #10001) 2 NDB nodes get signal 6 (abort) in DBTC
- (Bug #9969) 4012 - has misleading error message
- (Bug #9960) START BACKUP reports failure albeit succeeding
- (Bug #9924) ABORT BACKUP 1 crashes 4 node cluster
- (Bug #9892) Index activation file during node recovery
- (Bug #9891) Crash in DBACC (line 7004) during commit
- (Bug #9865) SELECT does not function properly
- (Bug #9839) Column with AUTOINC contains -1 Value on node stop

- (Bug #9757) Uncompleted node failure after gracefully stopping node
- (Bug #9749) Transactions causes deadlock in ACC
- (Bug #9724) Node fails to start: Message: File has already been opened
- (Bug #9691) UPDATE fails on attempt to update primary key
- (Bug #9675) Auto-increment not working with INSERT..SELECT and NDB storage
- (Bug #9318) drop database does not drop ndb tables
- (Bug #9280) Memory leak in cluster when dependent sub-queries are used
- (Bug #8928) create table with keys will shutdown the cluster
- Creating a table did not work for a cluster with 6 nodes. (Bug #8928)
Databases with 1, 2, 4, 8, ... (2^n nodes) did not have the problem. After a rolling upgrade, restart each node manually by restarting it with the --initial option. Otherwise, use dump and restore after an upgrade.

D.2.9. Changes in MySQL Cluster-4.1.11 (01 April 2005)

Functionality added or changed:

Bugs fixed:

- (Bug #9916) DbaccMain.cpp / DBACC (Line: 4876) / Pointer too large
- (Bug #9435) TIMESTAMP columns don't update
- (Bug #9052) Uninitialized data during unique index build, potential cluster crash
- (Bug #8876) Timeout when committing aborted transaction after node failure
- (Bug #8786) ndb_autodiscover, drop index can fail, wait 2 minutes timeout

- (Bug #8853) Transaction aborted after long time during node failure (4012)
- (Bug #8753) Invalid schema object version after dropping index (crash fixed, currently retry required)
- (Bug #8645) Assertion failure with multiple management servers
- (Bug #8557) ndbd does not get same nodeid on restart
- (Bug #8556) corrupt ndb_mgm show printout for certain configurations
- (Bug #8167) cluster shared memory and mysqld signal usage clash

D.2.10. Changes in MySQL Cluster-4.1.10 (12 February 2005)

Bugs fixed:

- (Bug #8284) Out of fragment memory in DBACC
- (Bug #8262) Node crash due to bug in DBLQH
- (Bug #8208) node restart fails on Aix 5.2
- (Bug #8167) cluster shared memory and mysqld signal usage clash
- (Bug #8101) unique index and error 4209 while selecting
- (Bug #8070) (Bug #7937) (Bug #6716) various ndb_restore core dumps on HP-UX
- (Bug #8010) 4006 forces MySQL Node Restart
- (Bug #7928) out of connection objects
- (Bug #7898) mysqld crash with ndb (solaris)
- (Bug #7864) Not possible to have more than 4.5G data memory

D.2.11. Changes in MySQL Cluster-4.1.9 (13 January 2005)

Functionality added or changed:

- New implementation of shared memory transporter.
- Cluster automatically configures shared memory transporter if possible.
- Cluster prioritizes usage of transporters with shared memory and localhost TCP
- Added switches to control the above functions, `ndb-shm` and `ndb-optimized-node-selection`.

Bugs fixed:

- (Bug #7805) config.ini parsing error
- (Bug #7798) Running range scan after alter table in different thread causes node failure
- (Bug #7761) Alter table does not autocommit
- (Bug #7725) Indexed DATETIME Columns Return Random Results
- (Bug #7660) START BACKUP does not increment BACKUP-ID (Big Endian machines)
- (Bug #7593) Cannot Create A Large NDB Data Warehouse
- (Bug #7480) Mysqld crash in ha_ndbcluster using Query Browser
- (Bug #7470) shared memory transporter does not connect
- (Bug #7396) Primary Key not working in NDB Mysql Clustered table (solaris)
- (Bug #7379) ndb restore fails to handle blobs and multiple databases
- (Bug #7346) ndb_restore enters infinite loop
- (Bug #7340) Problem for inserting data into the Text field on utf8

- (Bug #7124) `ndb_mgmd` is aborted on startup when using SHM connection

D.2.12. Changes in MySQL Cluster-4.1.8 (14 December 2004)

Functionality added or changed:

- Default port for **`ndb_mgmd`** was changed to 1186 (from 2200) as this port number was officially assigned to MySQL Cluster by IANA.
- New command in **`ndb_mgm`**, `PURGE STALE SESSIONS`, as a workaround for cases where nodes fail to allocate a node id even if it is free to use.
- New command in **`ndb_mgm`**, `CONNECT`.
- The `ndb` executables have been changed to make use of the regular MySQL command line option parsing features. See [Section 15.6.5, “Command Options for MySQL Cluster Processes”](#), for notes on changes.
- As bonus of the above you can now specify all command line options in `my.cnf` using the executable names as sections, that is, `[ndbd]`, `[ndb_mgmd]`, `[ndb_mgm]`, `[ndb_restore]`, and so forth.

```
[ndbd]
ndb-connectstring=myhost.domain.com:1234
[ndb_mgm]
ndb-connectstring=myhost.domain.com:1234
```

- Added use of section `[mysql_cluster]` in `my.cnf`. All cluster executables, including `mysqld`, parse this section. For example, this is a convenient place to put `ndb-connectstring` so that it need be specified only once.
- Added cluster log info events on allocation and deallocation of nodeid's.
- Added cluster log info events on connection refuse as a result of version mismatch.
- Extended `connectstring` syntax to allow for leaving the port number out. For example, `ndb-connectstring|connect-string=myhost1,myhost2,myhost3` is a valid `connectstring` and connect occurs on default port 1186.

- Clear text ndb error messages provided also for error codes that are mapped to corresponding mysql error codes, by executing SHOW WARNINGS after an error has occurred which relates to the ndb storage engine.
- Significant performance improvements done for read performance, especially for blobs.
- Added some variables for performance tuning, ndb_force_send and ndb_use_exact_count. Do show variables like 'ndb%'; in mysql client for listing. Use set command to alter variables.
- Added variables to set some options, ndb_use_transactions and ndb_autoincrement_prefetch_sz.

Bugs fixed:

- (Bug #7303) ndb_mgm: Trying to set CLUSTERLOG for a specific node id core dumps
- (Bug #7193) start backup gives false error printout
- (Bug #7153) Cluster nodes don't report error on endianness mismatch
- (Bug #7152) ndb_mgmd segmentation fault on incorrect HostName in configuration
- (Bug #7104) clusterlog filtering and level setting broken
- (Bug #6995) ndb_recover on varchar fields results in changing case of data
- (Bug #6919) all status only shows 2 nodes on a 8-node cluster
- (Bug #6871) DBD execute failed: Got error 897 'Unknown error code' from ndbcluster
- (Bug #6794) Wrong outcome of update operation of ndb table
- (Bug #6791) Segmentation fault when config.ini is not correctly set
- (Bug #6775) failure in acc when running many mysql clients

- (Bug #6696) `ndb_mgm` command line options inconsistent with behavior
- (Bug #6684) `ndb_restore` doesn't give error messages if improper command given
- (Bug #6677) `ndb_mgm` can crash on "ALL CLUSTERLOG"
- (Bug #6538) Error code returned when `select max()` on empty table with index
- (Bug #6451) failing create table gives "ghost" tables which are impossible to remove
- (Bug #6435) strange behavior of left join
- (Bug #6426) update with long pk fails
- (Bug #6398) update of primary key fails
- (Bug #6354) `mysql` does not complain about `--ndbcluster` option when NDB is not compiled in
- (Bug #6331) `INSERT IGNORE .. SELECT` breaks subsequent inserts
- (Bug #6288) cluster nodes crash on data import
- (Bug #6031) To drop database you have to execute `DROP DATABASE` command twice
- (Bug #6020) `LOCK TABLE + delete` returns error 208
- (Bug #6018) `REPLACE` does not work for BLOBs + NDB
- (Bug #6016) Strange crash with blobs + different DATABASES
- (Bug #5973) `ndb` table belonging to different database shows up in `show tables`
- (Bug #5872) `ALTER TABLE` with blob from `ndb` table to `myisam` fails
- (Bug #5844) Failing `mysql-test-run` leaves stray NDB processes behind

- (Bug #5824) HELP text messed up in ndb_mgm
- (Bug #5786) Duplicate key error after restore
- (Bug #5785) lock timeout during concurrent update
- (Bug #5782) Unknown error when using LIMIT with ndb table
- (Bug #5756) RESTART node from ndb_mgm fails
- A few more not reported bugs fixed

D.2.13. Changes in MySQL Cluster-4.1.7 (23 October 2004)

Functionality added or changed:

- Optimization 1: Improved performance on index scans. Measured 30% performance increase on query which do large amounts of index scans.
- Optimization 2: Improved performance on primary key lookups. Around double performance for autocommitted primary key lookups.
- Optimization 3: Improved performance when using blobs by avoiding usage of exclusive locks for blobs.

Bugs fixed:

- A few bugs fixed.

D.2.14. Changes in MySQL Cluster-4.1.6 (10 October 2004)

Functionality added or changed:

- Limited character set support for storage engine NDBCLUSTER:

Char set	Collation
big5	big5_chinese_ci
	big5_bin
binary	binary

euckr	euckr_korean_ci
	euckr_bin
gb2312	gb2312_chinese_ci
	gb2312_bin
gbk	gbk_chinese_ci
	gbk_bin
latin1	latin1_swedish_ci
	latin1_bin
sjis	sjis_japanese_ci
	sjis_bin
tis620	tis620_bin
ucs2	ucs2_general_ci
	ucs2_bin
ujis	ujis_japanese_ci
	ujis_bin
utf8	utf8_general_ci
	utf8_bin

- The SCI Transporter has been brought up-to-date with all changes and now works and has been documented as well.
- Optimizations when several clients to a MySQL Server access ndb tables.
- Added more checks and warnings for erroneous and inappropriate cluster configurations.
- SHOW TABLES now directly shows ndb tables created on a different MySQL server, that is, without a prior table access.
- Enhanced support for starting MySQL Server independently of ndbd and ndb_mgmd.
- Clear text ndb error messages provided by executing SHOW WARNINGS after an error has occurred which relates to the ndb storage engine.

Bugs fixed:

- Quite a few bugs fixed.

D.2.15. Changes in MySQL Cluster-4.1.5 (16 September 2004)

Functionality added or changed:

- Many queries in MySQL Cluster are executed as range scans or full table scans. All queries that don't use a unique hash index or the primary hash index use this access method. In a distributed system it is crucial that batching is properly performed.

In previous versions, the batch size was fixed to 16 per data node. In this version it is configurable per MySQL Server. So for queries using lots of large scans it is appropriate to set this parameter rather large and for queries using many small scans only fetching a small amount of records it is appropriate to set it low.

The performance of queries can easily change as much as 40% based on how this variable is set.

In future versions more logic will be implemented for assessing the batch size on a per-query basis. Thus, the semantics of the new configuration variable `ScanBatchSize` are likely to change.

- The fixed size overhead of the `ndbd` process has been greatly decreased. This is also true for the overhead per operation record as well as overhead per table and index.

A number of new configuration variables have been introduced to enable configuration of system buffers. Configuration variables for specifying the numbers of tables, unique hash indexes, and ordered indexes have also been introduced.

New configuration variables: `MaxNoOfOrderedIndexes`,
`MaxNoOfUniqueHashIndexes`

Configuration variables no longer used: `MaxNoOfIndexes` (split into the two above).

- In previous versions ALTER TABLE, TRUNCATE TABLE, and LOAD DATA were performed as one big transaction. In this version, all of these statements are automatically separated into several distinct transactions.

This removes the limitation that one could not change very large tables due to the MaxNoOfConcurrentOperations parameter.

- MySQL CLuster's online backup feature now backs up indexes so that both data and indexes are restored.
- In previous versions it was not possible to use NULL in indexes. This is now possible for all supported index types.
- Much work has been put onto making AUTO_INCREMENT features work as for other table handlers. Autoincrements as a partial key is still only supported by MyISAM.
- In earlier versions, **mysqld** would crash if the cluster wasn't started with the `--ndbcluster` option. Now **mysqld** handles cluster crashes and starts without crashing.
- The `-i` option for initial startup of **ndbd** has been removed. Initial startup still can be specified by using the `--initial` option. The reason for this is to ensure that it is clear what takes place when using `--initial`: this option completely removes all data from the disk and should only be used at initial start, in certain software upgrade cases, and in some cases as a workaround when nodes cannot be restarted successfully.
- The management client (**ndb_mgm**) now has additional commands and more information is printed for some commands such as `show`.
- In previous versions, the files were called `ndb_0..` when it wasn't possible to allocate a node ID when starting the node. To ensure that files are not so easily overwritten, these files are now named `ndb_pid..`, where `pid` is the process ID assigned by the OS.
- The default parameters have changed for **ndb_mgmd** and **ndbd**. In particular, they are now started as daemons by default. The `-n` option has been removed since it could cause confusion as to its meaning (`nostart` or `nodaemon`).

- In the configuration file, you can now use [NDBD] as an alias for [DB], [MYSQLD] as an alias for [API], and [NDB_MGMD] as an alias for [MGM]. **Note:** In fact, [NDBD], [MYSQLD], and [NDB_MGMD] are now the preferred designations, although the older ones will continue to be supported for some time to come in order to maintain backward compatibility.
- Many more checks for consistency in configuration have been introduced to in order to provide quicker feedback on configuration errors.
- In the connect string, it is now possible to use both ';' and ',' as the separator between entries. Thus, "nodeid=2,host=localhost:2200" is equivalent to "nodeid=2;host=localhost:2200".

In the configuration file, it is also possible to use ':' or '=' for assignment values. For example, `MaxNoOfOrderedIndexes : 128` and `MaxNoOfOrderedIndexes = 128` are equivalent expressions.

- The configuration variable names are now case insensitive, so `MaxNoOfOrderedIndexes: 128` is equivalent to `MAXNOOFORDEREDINDEXES = 128`.
- It is possible now to set the backup directory separately from the `FileSystemPath` by using the `BackupDir` configuration variable.

Log files and trace files can now be placed in any directory by setting the `DataDir` configuration variable.

`FileSystemPath` is no longer mandatory and defaults to `DataDir`.

- Queries involving tables from different databases are now supported.
- It is now possible to update the primary key.
- The performance of ordered indexes has been greatly improved, particularly the maintenance of indexes on updates, inserts and deletes.

Bugs fixed:

- Quite a few bugs fixed.

D.2.16. Changes in MySQL Cluster-4.1.4 (31 August 2004)

Functionality added or changed:

- The names of the log files and trace files created by the **ndbd** and **ndb_mgmd** processes have changed.
- Support for the many BLOB data types was introduced in this version.

Bugs fixed:

- Quite a few bugs were fixed in the 4.1.4 release.

D.2.17. Changes in MySQL Cluster-4.1.3 (28 June 2004)

Functionality added or changed:

- This was the first MySQL Cluster release so all functionality was new.

Bugs fixed:

- Various bugs fixed in the development process leading up to 4.1.3.

D.3. MySQL Connector/ODBC (MyODBC) Change History

D.3.1. Changes in MyODBC 3.51.13

Functionality added or changed:

- N/A

Bugs fixed:

- The `SQLDriverConnect()` ODBC method did not work with recent MyODBC releases. (Bug #12393)

D.3.2. Changes in MyODBC 3.51.12

Functionality added or changed:

- N/A

Bugs fixed:

- File DSNs could not be saved. (Bug #12019)
- `SQLColumns()` returned no information for tables that had a column named using a reserved word. (Bug #9539)

D.3.3. Changes in MyODBC 3.51.11

Functionality added or changed: No changes.

Bugs fixed:

- `mysql_list_dbcolumns()` and `insert_fields()` were retrieving all rows from a table. Fixed the queries generated by these functions to return no rows. (Bug #8198)
- `SQLGetTypeInfo()` returned `tinyblob` for `SQL_VARBINARY` and nothing for

SQL_BINARY. Fixed to return varbinary for SQL_VARBINARY, binary for SQL_BINARY, and longblob for SQL_LONGVARBINARY. (Bug #8138)

D.4. MySQL Connector/NET Change History

D.4.1. Version 1.0.8

- An exception would be raised when using an output parameter to a `System.String` value. (Bug #17814)
- The `DiscoverParameters` function would fail when a stored procedure used a `NUMERIC` parameter type. (Bug #19515)
- When running a query that included a date comparison, a `DateReader` error would be raised. (Bug #19481)
- Parameter substitution in queries where the order of parameters and table fields did not match would substitute incorrect values. (Bug #19261)
- When working with multiple threads, character set initialization would generate errors. (Bug #17106)
- When using an unsigned 64-bit integer in a stored procedure, the unsigned bit would be lost stored. (Bug #16934)
- The connection string parser did not allow single or double quotes in the password. (Bug #16659)
- The `CommandBuilder` ignored `Unsigned` flag at Parameter creation. (Bug #17375)
- `CHAR` type added to `MySqlDbType`. (Bug #17749)
- Unsigned data types were not properly supported. (Bug #16788)

D.4.2. Version 1.0.7

- The parameter collection object's `Add()` method added parameters to the list without first checking to see whether they already existed. Now it updates the value of the existing parameter object if it exists. (Bug #13927)

- A #42000Query was empty exception occurred when executing a query built with MySqlCommandBuilder, if the query string ended with a semicolon. (Bug #14631)
- Implemented the MySqlCommandBuilder.DeriveParameters method that is used to discover the parameters for a stored procedure. (Bug #13632)
- Added support for the cp932 character set. (Bug #13806)
- Calling a stored procedure where a parameter contained special characters (such as '@') would produce an exception. Note that ANSI_QUOTES had to be enabled to make this possible. (Bug #13753)
- A statement that contained multiple references to the same parameter could not be prepared. (Bug #13541)
- The Ping() method did not update the State property of the Connection object. (Bug #13658)

D.4.3. Version 1.0.6

- The nant build sequence had problems. (Bug #12978)
- Serializing a parameter failed if the first value passed in was NULL. (Bug #13276)
- Field names that contained the following characters caused errors: (%)%<>/ (Bug #13036)
- The MySQL Connector/NET 1.0.5 installer would not install alongside MySQL Connector/NET 1.0.4. (Bug #12835)
- MySQL Connector/NET 1.0.5 could not connect on Mono. (Bug #13345)

D.4.4. Version 1.0.5

- With multiple hosts in the connection string, MySQL Connector/NET would not connect to the last host in the list. (Bug #12628)
- MySQL Connector/NET interpreted the new decimal data type as a byte

array. (Bug #11294)

- The cp1250 character set was not supported. (Bug #11621)
- Connection could fail when .NET thread pool had no available worker threads. (Bug #10637)
- Decimal parameters caused syntax errors. (Bug #11550, Bug #10486, Bug #10152)
- A call to a stored procedure caused an exception if the stored procedure had no parameters. (Bug #11542)
- Certain malformed queries would trigger a Connection must be valid and open error message. (Bug #11490)
- The MySqlCommandBuilder class could not handle queries that referenced tables in a database other than the default database. (Bug #8382)
- MySQL Connector/NET could not work properly with certain regional settings. (WL#8228)
- Trying to use a stored procedure when Connection.Database was not populated generated an exception. (Bug #11450)
- Trying to read a TIMESTAMP column generated an exception. (Bug #7951)
- Parameters were not recognized when they were separated by linefeeds. (Bug #9722)
- Calling MySqlConnection.Clone when a connection string had not yet been set on the original connection would generate an error. (Bug #10281)
- Added support to call a stored function from MySQL Connector/NET. (Bug #10644)
- MySQL Connector/NET could not connect to MySQL 4.1.14. (Bug #12771)
- The ConnectionString property could not be set when a MySqlConnection

object was added with the designer. (Bug #12551, Bug #8724)

D.4.5. Version 1.0.4 1-20-05

- Bug #7243 calling prepare causing exception [fixed]
- Fixed another small problem with prepared statements
- Bug #7258 MySqlCommand.Connection returns an IDbConnection [fixed]
- Bug #7345 MySqlAdapter.Fill method throws Error message : Non-negative number required [fixed]
- Bug #7478 Clone method bug in MySqlCommand [fixed]
- Bug #7612 MySqlDataReader.GetString(index) returns non-Null value when field is Null [fixed]
- Bug #7755 MySqlReader.GetInt32 throws exception if column is unsigned [fixed]
- Bug #7704 GetBytes is working no more [fixed]
- Bug #7724 Quote character \222 not quoted in EscapeString [fixed]
- Fixed problem that causes named pipes to not work with some blob functionality
- Fixed problem with shared memory connections
- Bug #7436 Problem with Multiple resultsets... [fixed]
- Added or filled out several more topics in the API reference documentation

D.4.6. Version 1.0.3-gamma 12-10-04

- Made MySQL the default named pipe name
- Now SHOW COLLATION is used upon connection to retrieve the full list of charset ids

- Fixed Invalid character set index: 200 (Bug #6547)
- Installer now includes options to install into GAC and create Start Menu items
- Bug #6863 - Int64 Support in MySqlCommand Parameters [fixed]
- Connections now do not have to give a database on the connection string
- Bug #6770 - MySqlDataReader.GetChar(int i) throws IndexOutOfRangeException [fixed]
- Fixed problem where multiple resultsets having different numbers of columns would cause a problem
- Bug #6983 Exception stack trace lost when re-throwing exceptions [fixed]
- Fixed major problem with detecting null values when using prepared statements
- Bug #6902 Errors in parsing stored procedure parameters [fixed]
- Bug #6668 Integer "out" parameter from stored procedure returned as string [fixed]
- Bug #7032 MySqlDateTime in Datatables sorting by Text, not Date. [fixed]
- Bug #7133 Invalid query string when using inout parameters [fixed]
- Bug #6831 Test suite fails with MySQL 4.0 because of case sensitivity of table names [fixed]
- Bug #7132 Inserting DateTime causes System.InvalidCastException to be thrown [fixed]
- Bug #6879 InvalidCast when using DATE_ADD-function [fixed]
- Bug #6634 An Open Connection has been Closed by the Host System [fixed]
- Added ServerThread property to MySqlConnection to expose server thread

id

- Added Ping method to MySqlConnection
- Changed the name of the test suite to MySql.Data.Tests.dll

D.4.7. Version 1.0.2-gamma 04-11-15

- Fixed problem with MySqlBinary where string values could not be used to update extended text columns
- Fixed Installation directory ignored using custom installation (Bug #6329)
- Fixed problem where setting command text leaves the command in a prepared state
- Fixed double type handling in MySqlParameter(string parameterName, object value) (Bug #6428)
- Fixed Zero date "0000-00-00" is returned wrong when filling Dataset (Bug #6429)
- Fixed problem where calling stored procedures might cause an "Illegal mix of collations" problem.
- Added charset connection string option
- Fixed #HY000 Illegal mix of collations (latin1_swedish_ci,IMPLICIT) and (utf8_general_ (Bug #6322)
- Added the TableEditor CS and VB sample
- Fixed Charset-map for UCS-2 (Bug #6541)
- Updated the installer to include the new samples
- Fixed Long inserts take very long time (Bu #5453)
- Fixed Objects not being disposed (Bug #6649)

- Provider is now using character set specified by server as default

D.4.8. Version 1.0.1-beta2 04-10-27

- Fixed BUG #5602 Possible bug in MySqlConnection(string, object) constructor
- Fixed BUG #5458 Calling GetChars on a longtext column throws an exception
- Fixed BUG #5474 cannot run a stored procedure populating mysqlcommand.parameters
- Fixed BUG #5469 Setting DbType throws NullReferenceException
- Fixed problem where connector was not issuing a CMD_QUIT before closing the socket
- Fixed BUG #5392 MySqlCommand sees "?" as parameters in string literals
- Fixed problem with ConnectionInternal where a key might be added more than once
- CP1252 is now used for Latin1 only when the server is 4.1.2 and later
- Fixed BUG #5388 DataReader reports all rows as NULL if one row is NULL
- Virtualized driver subsystem so future releases could easily support client or embedded server support
- Field buffers being reused to decrease memory allocations and increase speed
- Fixed problem where using old syntax while using the interfaces caused problems
- Using PacketWriter instead of Packet for writing to streams
- Refactored compression code into CompressedStream to clean up

NativeDriver

- Added test case for resetting the command text on a prepared command
- Fixed problem where `MySqlParameterCollection.Add()` would throw unclear exception when given a null value (Bug #5621)
- Fixed constructor initialize problems in `MySqlCommand()` (Bug #5613)
- Fixed Parsing the ';' char (Bug #5876)
- Fixed missing Reference in `DbType` setter (Bug #5897)
- Fixed `System.OverflowException` when using `YEAR` datatype (Bug #6036)
- Added Aggregate function test (wasn't really a bug)
- Fixed serializing of floating point parameters (double, numeric, single, decimal) (Bug #5900)
- `IsNullable` error (Bug #5796)
- Fixed problem where connection lifetime on the connect string was not being respected
- Fixed problem where Min Pool Size was not being respected
- Fixed `MySqlDataReader` and 'show tables from ...' behavior (Bug #5256)
- Implemented `SequentialAccess`
- Fixed `MySqlDateTime` sets `IsZero` property on all `subseq.records` after first zero found (Bug #6006)
- Fixed Can't display Chinese correctly (Bug #5288)
- Fixed Russian character support as well
- Fixed `Method TokenizeSql()` uses only a limited set of valid characters for parameters (Bug #6217)

- Fixed NET Connector source missing resx files (Bug #6216)
- Fixed DBNull Values causing problems with retrieving/updating queries. (Bug #5798)
- Fixed Yet Another "object reference not set to an instance of an object" (Bug #5496)
- Fixed problem in PacketReader where it could try to allocate the wrong buffer size in EnsureCapacity
- Fixed GetBoolean returns wrong values (Bug #6227)
- Fixed IndexOutOfBounds when reading BLOB with DataReader with GetString(index) (Bug #6230)

D.4.9. Version 1.0.0 04-09-01

- Fixed BUG# 3889 Thai encoding not correctly supported
- Updated many of the test cases
- Fixed problem with using compression
- Bumped version number to 1.0.0 for beta 1 release
- Added COPYING.rtf file for use in installer
- Removed all of the XML comment warnings (I'll clean them up better later)
- Removed some last references to ByteFX

D.4.10. Version 0.9.0 04-08-30

- Added test fixture for prepared statements
- All type classes now implement a SerializeBinary method for sending their data to a PacketWriter
- Added PacketWriter class that will enable future low-memory large object

handling

- Fixed many small bugs in running prepared statements and stored procedures
- Changed command so that an exception will not be throw in executing a stored procedure with parameters in old syntax mode
- SingleRow behavior now working right even with limit
- GetBytes now only works on binary columns
- Logger now truncates long sql commands so blob columns don't blow out our log
- host and database now have a default value of "" unless otherwise set
- FIXED BUG# 5214 Connection Timeout seems to be ignored
- Added test case for bug# 5051: GetSchema not working correctly
- Fixed problem where GetSchema would return false for IsUnique when the column is key
- MySqlDataReader GetXXX methods now using the field level MySqlValue object and not performing conversions
- FIXED BUG# 5097: DataReader returning NULL for time column
- Added test case for LOAD DATA LOCAL INFILE
- Added replacetext custom nant task
- Added CommandBuilderTest fixture
- Added Last One Wins feature to CommandBuilder
- Fixed persist security info case problem
- Fixed GetBool so that 1, true, "true", and "yes" all count as trueWL# 2024
Make parameter mark configurable

- Added the "old syntax" connection string parameter to allow use of @ parameter marker
- Fixed Bug #4658 MySqlCommandBuilder
- Fixed Bug #4864 ByteFX.MySqlClient caches passwords if 'Persist Security Info' is false
- Updated license banner in all source files to include FLOSS exception
- Added new .Types namespace and implementations for most current MySql types
- Added MySqlField41 as a subclass of MySqlField
- Changed many classes to now use the new .Types types
- Changed type enum int to Int32, short to Int16, and bigint to Int64
- Added dummy types UInt16, UInt32, and UInt64 to allow an unsigned parameter to be made
- Connections are now reset when they are pulled from the connection pool
- Refactored auth code in driver so it can be used for both auth and reset
- Added UserReset test in PoolingTests.cs
- Connections are now reset using COM_CHANGE_USER when pulled from the pool
- Implemented SingleResultSet behavior
- Implemented support of unicode
- Added char set mappings for utf-8 and ucs-2
- fixed Bug #4520 time fields overflow using bytefx .net mysql driver
- Modified time test in data type test fixture to check for time spans where hours > 24

- Fixed Bug #4505 Wrong string with backslash escaping in ByteFx.Data.MySqlClient.MySqlParameter
- Added code to Parameter test case TestQuoting to test for backslashes
- Fixed Bug #4486 mysqlcommandbuilder fails with multi-word column names
- Fixed bug in TokenizeSql where underscore would terminate character capture in parameter name
- Added test case for spaces in column names
- Fixed bug# 4324 - MySqlDataReader.GetBytes don't works correctly
- Added GetBytes() test case to DataReader test fixture
- Now reading all server variables in InternalConnection.Configure into Hashtable
- Now using string[] for index map in CharSetMap
- Added CRInSQL test case for carriage returns in SQL
- setting maxPacketSize to default value in Driver.ctor
- Fixed bug #4442 - Setting MySqlDbType on a parameter doesn't set generic type
- Removed obsolete data types Long and LongLong
- Fixed bug# 4071 - Overflow exception thrown when using "use pipe" on connection string
- Changed "use pipe" keyword to "pipe name" or just "pipe"
- Allow reading multiple resultsets from a single query
- Added flags attribute to ServerStatusFlags enum
- Changed name of ServerStatus enum to ServerStatusFlags

- Fixed BUG #4386 - Inserted data row doesn't update properly
- Fixed bug #4074 - Error processing show create table
- Change Packet.ReadLenInteger to ReadPackedLong and added packet.ReadPackedInteger that always reads integers packed with 2,3,4
- Added syntax.cs test fixture to test various SQL syntax bugs
- Fixed bug# 4149 Improper handling of time values. Now time value of 00:00:00 is not treated as null.
- Moved all test suite files into TestSuite folder
- Fixed bug where null column would move the result packet pointer backward
- Added new nant build script
- Fixed BUG #3917 - clear tablename so it will be regen'ed properly during the next GenerateSchema.
- Fixed bug #3915 - GetValues was always returning zero and was also always trying to copy all fields rather than respecting the size of the array passed in.
- Implemented shared memory access protocol
- Implemented prepared statements for MySQL 4.1
- Implemented stored procedures for MySQL 5.0
- Renamed MySqlInternalConnection to InternalConnection
- SQL is now parsed as chars, fixes problems with other languages
- Added logging and allow batch connection string options
- Fixed bug #3888 - RowUpdating event not set when setting the DataAdapter property

- Fixed bug in char set mapping
- Implemented 4.1 authentication
- Improved open/auth code in driver
- Improved how connection bits are set during connection
- Database name is now passed to server during initial handshake
- Changed namespace for client to `MySql.Data.MySqlClient`
- Changed assembly name of client to `MySql.Data.dll`
- Changed license text in all source files to GPL
- Added the `MySqlClient.build` Nant file
- Removed the mono batch files
- Moved some of the unused files into `notused` folder so nant build file can use wildcards
- Implemented shared memory access
- Major revamp in code structure
- Prepared statements now working for `MySql 4.1.1` and later
- Finished implementing auth for `4.0`, `4.1.0`, and `4.1.1`
- Changed namespace from `MySQL.Data.MySQLClient` back to `MySql.Data.MySqlClient`
- Fixed bug in `CharSetMapping` where it was trying to use text names as ints
- Changed namespace to `MySQL.Data.MySQLClient`
- Integrated auth changes from `UC2004`
- Fixed bug where calling any of the `GetXXX` methods on a `datareader`

before or after reading data would not throw the appropriate exception (thanks Luca Morelli <morelli.luca@iol.it>)

- Added TimeSpan code in parameter.cs to properly serialize a timespan object to mysql time format (thanks Gianluca Colombo <g.colombo@alfi.it>)
- Added TimeStamp to parameter serialization code. Prevented DataAdapter updates from working right (thanks Michael King)
- Fixed a misspelling in MySqlHelper.cs (thanks Patrick Kristiansen)

D.4.11. Version 0.76

- Driver now using charset number given in handshake to create encoding
- Changed command editor to point to MySqlConnection.Design
- Fixed bug in Version.isAtLeast
- Changed DBConnectionString to support changes done to MySqlConnectionString
- Removed SqlCommandEditor and DataAdapterPreviewDialog
- Using new long return values in many places
- Integrated new CompressedStream class
- Changed ConnectionString and added attributes to allow it to be used in MySqlConnection.Design
- Changed packet.cs to support newer lengths in ReadLenInteger
- changed other classes to use new properties and fields of MySqlConnectionString
- ConnectionInternal is now using PING to see whether the server is alive
- Moved toolbox bitmaps into resource/

- Changed field.cs to allow values to come directly from row buffer
- Changed to use the new driver.Send syntax
- Using a new packet queueing system
- started work handling the "broken" compression packet handling
- Fixed bug in StreamCreator where failure to connect to a host would continue to loop infinitely (thanks Kevin Casella)
- Improved connectstring handling
- Moved designers into Pro product
- Removed some old commented out code from command.cs
- Fixed a problem with compression
- Fixed connection object where an exception throw prior to the connection opening would not leave the connection in the connecting state (thanks Chris Cline)
- Added GUID support
- Fixed sequence out of order bug (thanks Mark Reay)

D.4.12. Version 0.75

- Enum values now supported as parameter values (thanks Philipp Sumi)
- Year datatype now supported
- fixed compression
- Fixed bug where a parameter with a TimeSpan as the value would not serialize properly
- Fixed bug where default ctor would not set default connection string values
- Added some XML comments to some members

- Work to fix/improve compression handling
- Improved ConnectionString handling so that it better matches the standard set by SqlClient.
- A MySqlConnection is now thrown if a username is not included in the connection string
- Localhost is now used as the default if not specified on the connection string
- An exception is now thrown if an attempt is made to set the connection string while the connection is open
- Small changes to ConnectionString docs
- Removed MultiHostStream and MySqlStream. Replaced it with Common/StreamCreator
- Added support for Use Pipe connection string value
- Added Platform class for easier access to platform utility functions
- Fixed small pooling bug where new connection was not getting created after IsAlive fails
- Added Platform.cs and StreamCreator.cs
- Fixed Field.cs to properly handle 4.1 style timestamps
- Changed Common.Version to Common.DBVersion to avoid name conflict
- Fixed field.cs so that text columns return the right field type (thanks beni27@gmx.net)
- Added MySqlConnection class to provide some reference for error codes (thanks Geert Veenstra)

D.4.13. Version 0.74

- Added Unix socket support (thanks Mohammad DAMT [md@mt.web.id])
- only calling Thread.Sleep when no data is available
- improved escaping of quote characters in parameter data
- removed misleading comments from parameter.cs
- fixed pooling bug
- same pooling bug fixed again!! ;-)
- Fixed ConnectionString editor dialog (thanks marco p (pomarc))
- UserId now supported in connection strings (thanks Jeff Neeley)
- Attempting to create a parameter that is not input throws an exception (thanks Ryan Gregg)
- Added much documentation
- checked in new MultiHostStream capability. Big thanks to Dan Guisinger for this. he originally submitted the code and idea of supporting multiple machines on the connect string.
- Added alot of documentation. Still alot to do.
- Fixed speed issue with 0.73
- changed to Thread.Sleep(0) in MySqlDataStream to help optimize the case where it doesn't need to wait (thanks Todd German)
- Prepopulating the idlepools to MinPoolSize
- Fixed MySqlPool deadlock condition as well as stupid bug where CreateNewPooledConnection was not ever adding new connections to the pool. Also fixed MySqlStream.ReadBytes and ReadByte to not use TicksPerSecond which does not appear to always be right. (thanks Matthew J. Peddlesden)
- Fix for precision and scale (thanks Matthew J. Peddlesden)

- Added Thread.Sleep(1) to stream reading methods to be more cpu friendly (thanks Sean McGinnis)
- Fixed problem where ExecuteReader would sometime return null (thanks Lloyd Dupont)
- Fixed major bug with null field handling (thanks Naucki)
- enclosed queries for max_allowed_packet and charset inside try catch (and set defaults)
- fixed problem where socket was not getting closed properly (thanks Steve!)
- Fixed problem where ExecuteNonQuery was not always returning the right value
- Fixed InternalConnection to not use @@session.max_allowed_packet but use @@max_allowed_packet. (Thanks Miguel)
- Added many new XML doc lines
- Fixed sql parsing to not send empty queries (thanks Rory)
- Fixed problem where the reader was not unpeeking the packet on close
- Fixed problem where user variables were not being handled (thanks Sami Vaaraniemi)
- Fixed loop checking in the MySqlPool (thanks Steve M. Brown)
- Fixed ParameterCollection.Add method to match SqlClient (thanks Joshua Mouch)
- Fixed ConnectionString parsing to handle no and yes for boolean and not lowercase values (thanks Naucki)
- Added InternalConnection class, changes to pooling
- Implemented Persist Security Info
- Added security.cs and version.cs to project

- Fixed DateTime handling in Parameter.cs (thanks Burkhard Perken-Golomb)
- Fixed parameter serialization where some types would throw a cast exception
- Fixed DataReader to convert all returned values to prevent casting errors (thanks Keith Murray)
- Added code to Command.ExecuteReader to return null if the initial SQL command throws an exception (thanks Burkhard Perken-Golomb)
- Fixed ExecuteScalar bug introduced with restructure
- Restructure to allow for LOCAL DATA INFILE and better sequencing of packets
- Fixed several bugs related to restructure.
- Early work done to support more secure passwords in Mysql 4.1. Old passwords in 4.1 not supported yet
- Parameters appearing after system parameters are now handled correctly (Adam M. (adammil))
- strings can now be assigned directly to blob fields (Adam M.)
- Fixed float parameters (thanks Pent)
- Improved Parameter ctor and ParameterCollection.Add methods to better match SqlClient (thx Joshua Mouch)
- Corrected Connection.CreateCommand to return a MySqlCommand type
- Fixed connection string designer dialog box problem (thanks Abraham Guyt)
- Fixed problem with sending commands not always reading the response packet (thanks Joshua Mouch)
- Fixed parameter serialization where some blobs types were not being

handled (thanks Sean McGinnis)

- Removed spurious MessageBox.show from DataReader code (thanks Joshua Mouch)
- Fixed a nasty bug in the split sql code (thanks everyone! :-)

D.4.14. Version 0.71

- Fixed bug in MySqlStream where too much data could attempt to be read (thanks Peter Belbin)
- Implemented HasRows (thanks Nash Pherson)
- Fixed bug where tables with more than 252 columns cause an exception (thanks Joshua Kessler)
- Fixed bug where SQL statements ending in ; would cause a problem (thanks Shane Krueger)
- Fixed bug in driver where error messages were getting truncated by 1 character (thanks Shane Krueger)
- Made MySqlException serializable (thanks Mathias Hasselmann)

D.4.15. Version 0.70

- Updated some of the character code pages to be more accurate
- Fixed problem where readers could be opened on connections that had readers open
- Release of 0.70
- Moved test to separate assembly MySqlClientTests
- Fixed stupid problem in driver with sequence out of order (Thanks Peter Belbin)
- Added some pipe tests

- Increased default max pool size to 50
- Compiles with Mono 0-24
- Fixed connection and data reader dispose problems
- Added String datatype handling to parameter serialization
- Fixed sequence problem in driver that occurred after thrown exception (thanks Burkhard Perkens-Golomb)
- Added support for `CommandBehavior.SingleRow` to `DataReader`
- Fixed command sql processing so quotes are better handled (thanks Theo Spears)
- Fixed parsing of double, single, and decimal values to account for non-English separators. You still have to use the right syntax if you using hard coded sql, but if you use parameters the code will convert floating point types to use '.' appropriately internal both into the server and out. [Thanks anonymous]
- Added `MySQLStream` class to simplify timeOuts and driver coding.
- Fixed `DataReader` so that it is closed properly when the associated connection is closed. [thanks smishra]
- Made client more `SqlClient` compliant so that `DataReaders` have to be closed before the connection can be used to run another command
- Improved `DBNull.Value` handling in the fields
- Added several unit tests
- Fixed `MySQLException` so that the base class is actually called :-o
- Improved driver coding
- Fixed bug where `NextResult` was returning false on the last resultset
- Added more tests for MySQL

- Improved casting problems by equating unsigned 32bit values to Int64 and unsigned 16bit values to Int32, and so forth.
- Added new ctor for MySqlParameter for (name, type, size, srccol)
- Fixed bug in MySqlDataReader where it didn't check for null fieldlist before returning field count
- Started adding MySqlClient unit tests (added MySqlClient/Tests folder and some test cases)
- Fixed some things in Connection String handling
- Moved INIT_DB to MySqlPool. I may move it again, this is in preparation of the conference.
- Fixed bug inside CommandBuilder that prevented inserts from happening properly
- Reworked some of the internals so that all three execute methods of Command worked properly
- Fixed many small bugs found during benchmarking
- The first cut of ConnectionPooling is working. "min pool size" and "max pool size" are respected.
- Work to enable multiple resultsets to be returned
- Character sets are handled much more intelligently now. The driver queries MySQL at startup for the default character set. That character set is then used for conversions if that code page can be loaded. If not, then the default code page for the current OS is used.
- Added code to save the inferred type in the name,value ctor of Parameter
- Also, inferred type if value of null parameter is changed using Value property
- Converted all files to use proper Camel case. MySQL is now MySql in all

files. PgSQL is now PgSql

- Added attribute to PgSql code to prevent designer from trying to show
- Added MySQLDbType property to Parameter object and added proper conversion code to convert from DbType to MySQLDbType)
- Removed unused ObjectToString method from MySQLParameter.cs
- Fixed Add(..) method in ParameterCollection so that it doesn't use Add(name, value) instead.
- Fixed IndexOf and Contains in ParameterCollection to be aware that parameter names are now stored without @
- Fixed Command.ConvertSQLToBytes so it only allows characters that can be in MySQL variable names
- Fixed DataReader and Field so that blob fields read their data from Field.cs and GetBytes works right
- Added simple query builder editor to CommandText property of MySQLCommand
- Fixed CommandBuilder and Parameter serialization to account for Parameters not storing @ in their names
- Removed MySQLFieldType enum from Field.cs. Now using MySQLDbType enum
- Added Designer attribute to several classes to prevent designer view when using VS.Net
- Fixed Initial catalog typo in ConnectionString designer
- Removed 3 parameter ctor for MySQLParameter that conflicted with (name, type, value)
- changed MySQLParameter so paramName is now stored without leading @ (this fixed null inserts when using designer)

- Changed TypeConverter for MySQLParameter to use the ctor with all properties

D.4.16. Version 0.68

- Fixed sequence issue in driver
- Added DbParametersEditor to make parameter editing more like SqlClient
- Fixed Command class so that parameters can be edited using the designer
- Update connection string designer to support Use Compression flag
- Fixed string encoding so that European characters like ä will work correctly
- Creating base classes to aid in building new data providers
- Added support for UID key in connection string
- Field, parameter, command now using DBNull.Value instead of null
- CommandBuilder using DBNull.Value
- CommandBuilder now builds insert command correctly when an auto_insert field is not present
- Field now uses typeof keyword to return System.Types (performance)

D.4.17. Version 0.65

- MySQLCommandBuilder now implemented
- Transaction support now implemented (not all table types support this)
- GetSchemaTable fixed to not use xsd (for Mono)
- Driver is now Mono-compatible!!
- TIME data type now supported

- More work to improve Timestamp data type handling
- Changed signatures of all classes to match corresponding SqlClient classes

D.4.18. Version 0.60

- Protocol compression using SharpZipLib (www.icsharpcode.net)
- Named pipes on Windows now working properly
- Work done to improve Timestamp data type handling
- Implemented IEnumerable on DataReader so DataGrid would work

D.4.19. Version 0.50

- Speed increased dramatically by removing bugging network sync code
- Driver no longer buffers rows of data (more ADO.Net compliant)
- Conversion bugs related to TIMESTAMP and DATETIME fields fixed

D.5. MySQL Connector/J Change History

D.5.1. Changes in MySQL Connector/J 5.0.2-beta (11 July 2006)

- Fixed can't use XAConnection for local transactions when no global transaction is in progress. (fixes Bug#17401)
- Fixed driver fails on non-ASCII platforms. The driver was assuming that the platform character set would be a superset of MySQL's "latin1" when doing the handshake for authentication, and when reading error messages. We now use Cp1252 for all strings sent to the server during the handshake phase, and a hard-coded mapping of the "language" server variable to the character set that is used for error messages. (Fixes Bug#18086)
- Fixed ConnectionProperties (and thus some subclasses) are not serializable, even though some J2EE containers expect them to be. (Fixes Bug#19169)
- Fixed MySQLValidConnectionChecker for JBoss doesn't work with MySQLXADataSources. (Fixes Bug#20242)
- Better caching of character set converters (per-connection) to remove a bottleneck for multibyte character sets.
- Added connection/datasource property "pinGlobalTxToPhysicalConnection" (defaults to "false"). When set to "true", when using XAConnections, the driver ensures that operations on a given XID are always routed to the same physical connection. This allows the XAConnection to support "XA START ... JOIN" after "XA END" has been called, and is also a workaround for transaction managers that don't maintain thread affinity for a global transaction (most either always maintain thread affinity, or have it as a configuration option).
- MySQLXAConnection.recover(int flags) now allows combinations of XAResource.TMSTARTRSCAN and TMENDRSCAN. To simulate the "scanning" nature of the interface, we return all prepared XIDs for TMSTARTRSCAN, and no new XIDs for calls with TMNOFLAGS, or TMENDRSCAN when not in combination with TMSTARTRSCAN. This change was made for API

compliance, as well as integration with IBM WebSphere's transaction manager.

D.5.2. Changes in MySQL Connector/J 5.0.1-beta (Not Released)

Not released due to a packaging error

D.5.3. Changes in MySQL Connector/J 5.0.0-beta (22 December 2005)

- XDataSource implemented (ported from 3.2 branch which won't be released as a product). Use `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` as your datasource class name in your application server to utilize XA transactions in MySQL-5.0.10 and newer.
- `PreparedStatement.setString()` didn't work correctly when `sql_mode` on server contained `NO_BACKSLASH_ESCAPES` and no characters that needed escaping were present in the string.
- Attempt detection of the MySQL type `BINARY` (it's an alias, so this isn't always reliable), and use the `java.sql.Types.BINARY` type mapping for it.
- Moved `-bin-g.jar` file into separate debug subdirectory to avoid confusion.
- Don't allow `.setAutoCommit(true)`, or `.commit()` or `.rollback()` on an XA-managed connection as per the JDBC specification.
- If the connection `useTimezone` is set to `true`, then also respect time zone conversions in escape-processed string literals (for example, `"{ts ...}"` and `"{t ...}"`).
- Return original column name for `RSMD.getColumnname()` if the column was aliased, alias name for `.getColumnLabel()` (if aliased), and original table name for `.getTableName()`. Note this only works for MySQL-4.1 and newer, as older servers don't make this information available to clients.
- Setting `useJDBCCompliantTimezoneShift=true` (it's not the default) causes

the driver to use GMT for *all* `TIMESTAMP/DATETIME` time zones, and the current VM time zone for any other type that refers to time zones. This feature can not be used when `useTimezone=true` to convert between server and client time zones.

- Add one level of indirection of internal representation of `CallableStatement` parameter metadata to avoid class not found issues on JDK-1.3 for `ParameterMetadata` interface (which doesn't exist prior to JDBC-3.0).
- Added unit tests for `XADatasource`, as well as friendlier exceptions for XA failures compared to the "stock" `XAException` (which has no messages).
- Idle timeouts cause `XAConnections` to whine about rolling themselves back. (Bug #14729)
- Added support for Connector/MXJ integration via url subprotocol `jdbc:mysql:mxj://....`
- Moved all `SQLException` constructor usage to a factory in `SQLException` (ground-work for JDBC-4.0 `SQLState`-based exception classes).
- Removed Java5-specific calls to `BigDecimal` constructor (when result set value is ' ', `(int)0` was being used as an argument indirectly via method return value. This signature doesn't exist prior to Java5.)
- Added service-provider entry to `META-INF/services/java.sql.Driver` for JDBC-4.0 support.
- Return "[VAR]BINARY" for `RSMD.getColumnTypeName()` when that is actually the type, and it can be distinguished (MySQL-4.1 and newer).
- When fix for Bug #14562 was merged from 3.1.12, added functionality for `CallableStatement`'s parameter metadata to return correct information for `.getParameterClassName()`.
- Fuller synchronization of `Connection` to avoid deadlocks when using multithreaded frameworks that multithread a single connection (usually not recommended, but the JDBC spec allows it anyways), part of fix to Bug #14972).

- Implementation of `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL-5.0.0 or newer server, require a separate connection to issue the `KILL QUERY` statement, and in the case of `setQueryTimeout()` creates an additional thread to handle the timeout functionality.

Note: Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeExceptions` rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

D.5.4. Changes in MySQL Connector/J 3.1.14 (not yet released)

- Fixed updatable result set throws `ClassCastException` when there is row data and `moveToInsertRow()` is called. (Fixes Bug#20479)
- Fixed Updatable result set that contains a BIT column fails when server-side prepared statements are used. (Fixes Bug#20485)
- Fixed memory leak with `profileSQL=true`. (Fixes Bug#16987)
- - - Connection fails to localhost when using timeout and IPv6 is configured. (Fixes Bug#19726)
- Fixed `NullPointerException` in `MysqlDataSourceFactory` due to Reference containing `RefAdrs` with null content. (Fixes Bug#16791)
- Fixed `ResultSet.getShort()` for UNSIGNED TINYINT returns incorrect values when using server-side prepared statements. (Fixes Bug#20306)
- Fixed can't pool server-side prepared statements, exception raised when re-using them. (Fixes Bug#20687 -

D.5.5. Changes in MySQL Connector/J 3.1.13 (26 May 2006)

- INOUT parameter does not store IN value. (Bug #15464)
- Exception thrown for new decimal type when using updatable result sets.

(Bug #14609)

- No "dos" character set in MySQL > 4.1.0. (Bug #15544)
- `PreparedStatement.setObject()` serializes `BigInteger` as object, rather than sending as numeric value (and is thus not complementary to `.getObject()` on an `UNSIGNED LONG` type). (Bug #15383)
- `ResultSet.getShort()` for `UNSIGNED TINYINT` returned wrong values. (Bug #11874)
- `lib-nodist` directory missing from package breaks out-of-box build. (Bug #15676)
- `DBMD.getColumns()` returns wrong type for `BIT`. (Bug #15854)
- Fixed issue where driver was unable to initialize character set mapping tables. Removed reliance on `.properties` files to hold this information, as it turns out to be too problematic to code around class loader hierarchies that change depending on how an application is deployed. Moved information back into the `CharsetMapping` class. (Bug #14938)
- Fixed updatable result set doesn't return `AUTO_INCREMENT` values for `insertRow()` when multiple column primary keys are used. (the driver was checking for the existence of single-column primary keys and an autoincrement value > 0 instead of a straightforward `isAutoIncrement()` check). (Bug #16841)
- Fixed `Statement.getGeneratedKeys()` throws `NullPointerException` when no query has been processed. (Bug #17099)
- Fixed driver trying to call methods that don't exist on older and newer versions of `Log4j`. The fix is not trying to auto-detect presence of `log4j`, too many different incompatible versions out there in the wild to do this reliably. (Bug #13469)

If you relied on autodetection before, you will need to add `"logger=com.mysql.jdbc.log.Log4JLogger"` to your JDBC URL to enable `Log4J` usage, or alternatively use the new `"CommonsLogger"` class to take care of this.

- Added support for Apache Commons logging, use "com.mysql.jdbc.log.CommonsLogger" as the value for the "logger" configuration property.
- LogFactory now prepends "com.mysql.jdbc.log" to log class name if it can't be found as-specified. This allows you to use "short names" for the built-in log factories, for example "logger=CommonsLogger" instead of "logger=com.mysql.jdbc.log.CommonsLogger".
- Fixed issue with ReplicationConnection incorrectly copying state, doesn't transfer connection context correctly when transitioning between the same read-only states. (Bug #15570)
- Fixed issue where server-side prepared statements don't cause truncation exceptions to be thrown when truncation happens. (Bug #18041)
- Added performance feature, re-writing of batched executes for Statement.executeBatch() (for all DML statements) and PreparedStatement.executeBatch() (for INSERTs with VALUE clauses only). Enable by using "rewriteBatchedStatements=true" in your JDBC URL.
- Fixed CallableStatement.registerOutParameter() not working when some parameters pre-populated. Still waiting for feedback from JDBC experts group to determine what correct parameter count from getMetaData() should be, however. (Bug #17898)
- Fixed calling clearParameters() on a closed prepared statement causes NPE. (Bug #17587)
- Map "latin1" on MySQL server to CP1252 for MySQL > 4.1.0.
- Added additional accessor and mutator methods on ConnectionProperties so that DataSource users can use same naming as regular URL properties.
- Fixed data truncation and getWarnings() only returns last warning in set. (Bug #18740)
- Improved performance of retrieving BigDecimal, Time, Timestamp and Date values from server-side prepared statements by creating fewer short-

lived instances of Strings when the native type is not an exact match for the requested type. Fixes Bug #18496 for BigDecimals.

- Fixed aliased column names where length of name > 251 are corrupted. (Bug #18554)
- Fixed `ResultSet.isNull()` not always reset correctly for booleans when done via conversion for server-side prepared statements. (Bug #17450)
- Fixed invalid classname returned for `ResultSetMetaData.getColumnClassName()` for `BIGINT` type. (Bug #19282)
- Fixed case where driver wasn't reading server status correctly when fetching server-side prepared statement rows, which in some cases could cause warning counts to be off, or multiple result sets to not be read off the wire.
- Driver now aware of fix for `BIT` type metadata that went into MySQL-5.0.21 for server not reporting length consistently (Bug #13601).
- Fixed `PreparedStatement.setObject(int, Object, int)` doesn't respect scale of `BigDecimals`. (Bug #19615)
- Fixed `ResultSet.isNull()` returns incorrect value when extracting native string from server-side prepared statement generated result set. (Bug #19282)

D.5.6. Changes in MySQL Connector/J 3.1.12 (30 November 2005)

- Fixed client-side prepared statement bug with embedded `?` characters inside quoted identifiers (it was recognized as a placeholder, when it was not).
- Don't allow `executeBatch()` for `CallableStatements` with registered `OUT/INOUT` parameters (JDBC compliance).
- Fall back to platform-encoding for `URLDecoder.decode()` when parsing driver URL properties if the platform doesn't have a two-argument version of this method.

- Java type conversion may be incorrect for MEDIUMINT. (Bug #14562)
- Added configuration property `useGmtMillisForDatetimes` which when set to true causes `ResultSet.getDate()`, `ResultSet.getTimestamp()` to return correct millis-since GMT when `ResultSet.getTime()` is called on the return value (currently default is false for legacy behavior).
- Fixed `DatabaseMetaData.stores*Identifiers()`:
 - If `lower_case_table_names=0` (on server):
 - `storesLowerCaseIdentifiers()` returns false
 - `storesLowerCaseQuotedIdentifiers()` returns false
 - `storesMixedCaseIdentifiers()` returns true
 - `storesMixedCaseQuotedIdentifiers()` returns true
 - `storesUpperCaseIdentifiers()` returns false
 - `storesUpperCaseQuotedIdentifiers()` returns true
 - If `lower_case_table_names=1` (on server):
 - `storesLowerCaseIdentifiers()` returns true
 - `storesLowerCaseQuotedIdentifiers()` returns true
 - `storesMixedCaseIdentifiers()` returns false
 - `storesMixedCaseQuotedIdentifiers()` returns false
 - `storesUpperCaseIdentifiers()` returns false
 - `storesUpperCaseQuotedIdentifiers()` returns true
- `DatabaseMetaData.getColumns()` doesn't return `TABLE_NAME` correctly. (Bug #14815)
- Escape processor replaces quote character in quoted string with string

delimiter. (Bug #14909)

- OpenOffice expects `DBMD.supportsIntegrityEnhancementFacility()` to return true if foreign keys are supported by the datasource, even though this method also covers support for check constraints, which MySQL *doesn't* have. Setting the configuration property `overrideSupportsIntegrityEnhancementFacility` to true causes the driver to return true for this method. (Bug #12975)
- Added `com.mysql.jdbc.testsuite.url.default` system property to set default JDBC url for testsuite (to speed up bug resolution when I'm working in Eclipse).
- Unable to initialize character set mapping tables (due to J2EE classloader differences). (Bug #14938)
- Deadlock while closing server-side prepared statements from multiple threads sharing one connection. (Bug #14972)
- `logSlowQueries` should give better info. (Bug #12230)
- Extraneous sleep on `autoReconnect`. (Bug #13775)
- Driver incorrectly closes streams passed as arguments to `PreparedStatement`. Reverts to legacy behavior by setting the JDBC configuration property `autoClosePstmtStreams` to true (also included in the 3-0-Compat configuration "bundle"). (Bug #15024)
- `maxQuerySizeToLog` is not respected. Added logging of bound values for `execute()` phase of server-side prepared statements when `profileSQL=true` as well. (Bug #13048)
- Usage advisor complains about unreferenced columns, even though they've been referenced. (Bug #15065)
- Don't increase timeout for failover/reconnect. (Bug #6577)
- Process escape tokens in `Connection.prepareStatement(...)`. (Bug #15141) You can disable this behavior by setting the JDBC URL configuration property `processEscapeCodesForPrepStmts` to false.

- Reconnect during middle of `executeBatch()` should not occur if `autoReconnect` is enabled. (Bug #13255)

D.5.7. Changes in MySQL Connector/J 3.1.11-stable (07 October 2005)

- Spurious ! on console when character encoding is `utf8`. (Bug #11629)
- Fixed statements generated for testcases missing ; for “plain” statements.
- Incorrect generation of testcase scripts for server-side prepared statements. (Bug #11663)
- Fixed regression caused by fix for Bug #11552 that caused driver to return incorrect values for unsigned integers when those integers were within the range of the positive signed type.
- Moved source code to Subversion repository.
- Escape tokenizer doesn't respect stacked single quotes for escapes. (Bug #11797)
- `GEOMETRY` type not recognized when using server-side prepared statements.
- `ReplicationConnection` won't switch to slave, throws “Catalog can't be null” exception. (Bug #11879)
- Properties shared between master and slave with replication connection. (Bug #12218)
- `Statement.getWarnings()` fails with NPE if statement has been closed. (Bug #10630)
- Only `getChar[]` from SQL in `PreparedStatement.ParseInfo()` when needed.
- Geometry types not handled with server-side prepared statements. (Bug #12104)
- `StringUtils.getBytes()` doesn't work when using multi-byte character

encodings and a length in *characters* is specified. (Bug #11614)

- `Pstmt.setObject(..., Types.BOOLEAN)` throws exception. (Bug #11798)
- `maxPerformance.properties` mis-spells “`elideSetAutoCommits`”. (Bug #11976)
- `DBMD.storesLower/Mixed/UpperIdentifiers()` reports incorrect values for servers deployed on Windows. (Bug #11575)
- `ResultSet.moveToCurrentRow()` fails to work when preceded by a call to `ResultSet.moveToInsertRow()`. (Bug #11190)
- `VARBINARY` data corrupted when using server-side prepared statements and `.setBytes()`. (Bug #11115)
- `explainSlowQueries` hangs with server-side prepared statements. (Bug #12229)
- Escape processor didn't honor strings demarcated with double quotes. (Bug #11498)
- Lifted restriction of changing streaming parameters with server-side prepared statements. As long as all streaming parameters were set before execution, `.clearParameters()` does not have to be called. (due to limitation of client/server protocol, prepared statements can not reset *individual* stream data on the server side).
- Reworked `Field` class, `*Buffer`, and `MysqlIO` to be aware of field lengths $> Integer.MAX_VALUE$.
- Updated `DBMD.supportsCorrelatedQueries()` to return true for versions > 4.1 , `supportsGroupByUnrelated()` to return true and `getResultSetHoldability()` to return `HOLD_CURSORS_OVER_COMMIT`.
- Handling of catalog argument in `DatabaseMetaData.getIndexInfo()`, which also means changes to the following methods in `DatabaseMetaData`: (Bug #12541)
 - `getBestRowIdentifier()`

- `getColumns()`
- `getCrossReference()`
- `getExportedKeys()`
- `getImportedKeys()`
- `getIndexInfo()`
- `getPrimaryKeys()`
- `getProcedures()` (and thus indirectly `getProcedureColumns()`)
- `getTables()`

The catalog argument in all of these methods now behaves in the following way:

- Specifying `NULL` means that catalog will not be used to filter the results (thus all databases will be searched), unless you've set `nullCatalogMeansCurrent=true` in your JDBC URL properties.
- Specifying `""` means “current” catalog, even though this isn't quite JDBC spec compliant, it's there for legacy users.
- Specifying a catalog works as stated in the API docs.
- Made `Connection.clientPrepare()` available from “wrapped” connections in the `jdbc2.optional` package (connections built by `ConnectionPoolDataSource` instances).
- Added `Connection.isMasterConnection()` for clients to be able to determine if a multi-host master/slave connection is connected to the first host in the list.
- Tokenizer for `=` in URL properties was causing `sessionVariables=...` to be parameterized incorrectly. (Bug #12753)
- Foreign key information that is quoted is parsed incorrectly when `DatabaseMetaData` methods use that information. (Bug #11781)

- The `sendBlobChunkSize` property is now clamped to `max_allowed_packet` with consideration of stream buffer size and packet headers to avoid `PacketTooBigExceptions` when `max_allowed_packet` is similar in size to the default `sendBlobChunkSize` which is 1M.
- `CallableStatement.clearParameters()` now clears resources associated with `INOUT/OUTPUT` parameters as well as `INPUT` parameters.
- `Connection.prepareCall()` is database name case-sensitive (on Windows systems). (Bug #12417)
- `cp1251` incorrectly mapped to `win1251` for servers newer than 4.0.x. (Bug #12752)
- `java.sql.Types.OTHER` returned for `BINARY` and `VARBINARY` columns when using `DatabaseMetaData.getColumns()`. (Bug #12970)
- `ServerPreparedStatement.getBinding()` now checks if the statement is closed before attempting to reference the list of parameter bindings, to avoid throwing a `NullPointerException`.
- `ResultSetMetaData` from `Statement.getGeneratedKeys()` caused a `NullPointerException` to be thrown whenever a method that required a connection reference was called. (Bug #13277)
- Backport of `Field` class, `ResultSetMetaData.getColumnClassName()`, and `ResultSet.getObject(int)` changes from 5.0 branch to fix behavior surrounding `VARCHAR BINARY/VARBINARY` and related types.
- Fixed `NullPointerException` when converting catalog parameter in many `DatabaseMetaDataMethods` to `byte[]`s (for the result set) when the parameter is `null`. (`null` isn't technically allowed by the JDBC specification, but we've historically allowed it).
- Backport of `VAR[BINARY|CHAR] [BINARY]` types detection from 5.0 branch.
- Read response in `MysqlIO.sendFileToServer()`, even if the local file can't be opened, otherwise next query issued will fail, because it's reading the response to the empty `LOAD DATA INFILE` packet sent to the server.

- Workaround for Bug #13374: `ResultSet.getStatement()` on closed result set returns `NULL` (as per JDBC 4.0 spec, but not backward-compatible). Set the connection property `retainStatementAfterResultSetClose` to `true` to be able to retrieve a `ResultSet`'s statement after the `ResultSet` has been closed via `.getStatement()` (the default is `false`, to be JDBC-compliant and to reduce the chance that code using JDBC leaks `Statement` instances).
- URL configuration parameters don't allow `'&'` or `'='` in their values. The JDBC driver now parses configuration parameters as if they are encoded using the `application/x-www-form-urlencoded` format as specified by `java.net.URLDecoder` (<http://java.sun.com/j2se/1.5.0/docs/api/java/net/URLDecoder.html>). (Bug #13453)

If the `'%'` character is present in a configuration property, it must now be represented as `%25`, which is the encoded form of `'%'` when using `application/x-www-form-urlencoded` encoding.

- The configuration property `sessionVariables` now allows you to specify variables that start with the `'@'` sign.
- When `gatherPerfMetrics` is enabled for servers older than 4.1.0, a `NullPointerException` is thrown from the constructor of `ResultSet` if the query doesn't use any tables. (Bug #13043)

D.5.8. Changes in MySQL Connector/J 3.1.10-stable (23 June 2005)

- Fixed connecting without a database specified raised an exception in `MySQLIO.changeDatabaseTo()`.
- Initial implementation of `ParameterMetadata` for `PreparedStatement.getParameterMetadata()`. Only works fully for `CallableStatements`, as current server-side prepared statements return every parameter as a `VARCHAR` type.

D.5.9. Changes in MySQL Connector/J 3.1.9-stable (22 June 2005)

- Overhaul of character set configuration, everything now lives in a

properties file.

- Driver now correctly uses CP932 if available on the server for Windows-31J, CP932 and MS932 java encoding names, otherwise it resorts to SJIS, which is only a close approximation. Currently only MySQL-5.0.3 and newer (and MySQL-4.1.12 or .13, depending on when the character set gets backported) can reliably support any variant of CP932.
- `com.mysql.jdbc.PreparedStatement.ParseInfo` does unnecessary call to `toCharArray()`. (Bug #9064)
- Memory leak in `ServerPreparedStatement` if `serverPrepare()` fails. (Bug #10144)
- Actually write manifest file to correct place so it ends up in the binary jar file.
- Added `createDatabaseIfNotExist` property (default is `false`), which will cause the driver to ask the server to create the database specified in the URL if it doesn't exist. You must have the appropriate privileges for database creation for this to work.
- Unsigned `SMALLINT` treated as signed for `ResultSet.getInt()`, fixed all cases for `UNSIGNED` integer values and server-side prepared statements, as well as `ResultSet.getObject()` for `UNSIGNED TINYINT`. (Bug #10156)
- Double quotes not recognized when parsing client-side prepared statements. (Bug #10155)
- Made `enableStreamingResults()` visible on `com.mysql.jdbc.jdbc2.optional.StatementWrapper`.
- Made `ServerPreparedStatement.asSql()` work correctly so auto-explain functionality would work with server-side prepared statements.
- Made JDBC2-compliant wrappers public in order to allow access to vendor extensions.
- Cleaned up logging of profiler events, moved code to dump a profiler event as a string to `com.mysql.jdbc.log.LogUtils` so that third parties can use it.

- `DatabaseMetaData.supportsMultipleOpenResults()` now returns `true`. The driver has supported this for some time, DBMD just missed that fact.
- Driver doesn't support `{?=CALL(...)}` for calling stored functions. This involved adding support for function retrieval to `DatabaseMetaData.getProcedures()` and `getProcedureColumns()` as well. (Bug #10310)
- `SQLException` thrown when retrieving `YEAR(2)` with `ResultSet.getString()`. The driver will now always treat `YEAR` types as `java.sql.Date`s and return the correct values for `getString()`. Alternatively, the `yearIsDateType` connection property can be set to `false` and the values will be treated as `SHORT`s. (Bug #10485)
- The datatype returned for `TINYINT(1)` columns when `tinyInt1isBit=true` (the default) can be switched between `Types.BOOLEAN` and `Types.BIT` using the new configuration property `transformedBitIsBoolean`, which defaults to `false`. If set to `false` (the default), `DatabaseMetaData.getColumns()` and `ResultSetMetaData.getColumnType()` will return `Types.BOOLEAN` for `TINYINT(1)` columns. If `true`, `Types.BIT` will be returned instead. Regardless of this configuration property, if `tinyInt1isBit` is enabled, columns with the type `TINYINT(1)` will be returned as `java.lang.Boolean` instances from `ResultSet.getObject(...)`, and `ResultSetMetaData.getColumnClassName()` will return `java.lang.Boolean`.
- `SQLException` is thrown when using property `characterSetResults` with `cp932` or `eucjms`. (Bug #10496)
- Reorganized directory layout. Sources now are in `src` folder. Don't pollute parent directory when building, now output goes to `./build`, distribution goes to `./dist`.
- Added support/bug hunting feature that generates `.sql` test scripts to `STDERR` when `autoGenerateTestcaseScript` is set to `true`.
- 0-length streams not sent to server when using server-side prepared statements. (Bug #10850)
- Setting `cachePrepStmts=true` now causes the `Connection` to also cache the

check the driver performs to determine if a prepared statement can be server-side or not, as well as caches server-side prepared statements for the lifetime of a connection. As before, the `prepStmtCacheSize` parameter controls the size of these caches.

- Try to handle `OutOfMemoryErrors` more gracefully. Although not much can be done, they will in most cases close the connection they happened on so that further operations don't run into a connection in some unknown state. When an OOM has happened, any further operations on the connection will fail with a "Connection closed" exception that will also list the OOM exception as the reason for the implicit connection close event.
- Don't send `COM_RESET_STMT` for each execution of a server-side prepared statement if it isn't required.
- Driver detects if you're running MySQL-5.0.7 or later, and does not scan for `LIMIT ?[, ?]` in statements being prepared, as the server supports those types of queries now.
- `VARBINARY` data corrupted when using server-side prepared statements and `ResultSet.getBytes()`. (Bug #11115)
- `Connection.setCatalog()` is now aware of the `useLocalSessionState` configuration property, which when set to `true` will prevent the driver from sending `USE . . .` to the server if the requested catalog is the same as the current catalog.
- Added the following configuration bundles, use one or many via the `useConfigs` configuration property:
 - `maxPerformance` — maximum performance without being reckless
 - `solarisMaxPerformance` — maximum performance for Solaris, avoids syscalls where it can
 - `3-0-Compat` — Compatibility with Connector/J 3.0.x functionality
- Added `maintainTimeStats` configuration property (defaults to `true`), which tells the driver whether or not to keep track of the last query time and the last successful packet sent to the server's time. If set to `false`, removes

two syscalls per query.

- `autoReconnect` ping causes exception on connection startup. (Bug #11259)
- Connector/J dumping query into `SQLException` twice. (Bug #11360)
- Fixed `PreparedStatement.setClob()` not accepting `null` as a parameter.
- Production package doesn't include JBoss integration classes. (Bug #11411)
- Removed nonsensical “costly type conversion” warnings when using usage advisor.

D.5.10. Changes in MySQL Connector/J 3.1.8-stable (14 April 2005)

- Fixed `DatabaseMetaData.getTables()` returning views when they were not asked for as one of the requested table types.
- Added support for new precision-math `DECIMAL` type in MySQL 5.0.3 and up.
- Fixed `ResultSet.getTime()` on a `NULL` value for server-side prepared statements throws `NPE`.
- Made `Connection.ping()` a public method.
- `DATE_FORMAT()` queries returned as `BLOBs` from `getObject()`. (Bug #8868)
- `ServerPreparedStatements` now correctly “stream” `BLOB/CLOB` data to the server. You can configure the threshold chunk size using the JDBC URL property `blobSendChunkSize` (the default is 1MB).
- `BlobFromLocator` now uses correct identifier quoting when generating prepared statements.
- Server-side session variables can be preset at connection time by passing them as a comma-delimited list for the connection property `sessionVariables`.

- Fixed regression in `ping()` for users using `autoReconnect=true`.
- `PreparedStatement.addBatch()` doesn't work with server-side prepared statements and streaming BINARY data. (Bug #9040)
- `DBMD.supportsMixedCase*Identifiers()` returns wrong value on servers running on case-sensitive filesystems. (Bug #8800)
- Cannot use UTF-8 for `characterSetResults` configuration property. (Bug #9206)
- A continuation of Bug #8868, where functions used in queries that should return non-string types when resolved by temporary tables suddenly become opaque binary strings (work-around for server limitation). Also fixed fields with type of CHAR(n) CHARACTER SET BINARY to return correct/matching classes for `RSMD.getColumnClassName()` and `ResultSet.getObject()`. (Bug #9236)
- `DBMD.supportsResultSetConcurrency()` not returning true for forward-only/read-only result sets (we obviously support this). (Bug #8792)
- `DATA_TYPE` column from `DBMD.getBestRowIdentifier()` causes `ArrayIndexOutOfBoundsException` when accessed (and in fact, didn't return any value). (Bug #8803)
- Check for empty strings (' ') when converting CHAR/VARCHAR column data to numbers, throw exception if `emptyStringsConvertToZero` configuration property is set to false (for backward-compatibility with 3.0, it is now set to true by default, but will most likely default to false in 3.2).
- `PreparedStatement.getMetaData()` inserts blank row in database under certain conditions when not using server-side prepared statements. (Bug #9320)
- `Connection.canHandleAsPreparedStatement()` now makes “best effort” to distinguish LIMIT clauses with placeholders in them from ones without in order to have fewer false positives when generating work-arounds for statements the server cannot currently handle as server-side prepared statements.

- Fixed `build.xml` to not compile `log4j` logging if `log4j` not available.
- Added support for the `c3p0` connection pool's (<http://c3p0.sf.net/>) validation/connection checker interface which uses the lightweight `COM_PING` call to the server if available. To use it, configure your `c3p0` connection pool's `connectionTesterClassName` property to use `com.mysql.jdbc.integration.c3p0.MysqlConnectionTester`.
- Better detection of `LIMIT` inside/outside of quoted strings so that the driver can more correctly determine whether a prepared statement can be prepared on the server or not.
- Stored procedures with same name in different databases confuse the driver when it tries to determine parameter counts/types. (Bug #9319)
- Added finalizers to `ResultSet` and `Statement` implementations to be JDBC spec-compliant, which requires that if not explicitly closed, these resources should be closed upon garbage collection.
- Stored procedures with `DECIMAL` parameters with storage specifications that contained `' , '` in them would fail. (Bug #9682)
- `PreparedStatement.setObject(int, Object, int type, int scale)` now uses `scale` value for `BigDecimal` instances.
- `Statement.getMoreResults()` could throw `NPE` when existing result set was `.close()`d. (Bug #9704)
- The performance metrics feature now gathers information about number of tables referenced in a `SELECT`.
- The logging system is now automatically configured. If the value has been set by the user, via the URL property `logger` or the system property `com.mysql.jdbc.logger`, then use that, otherwise, autodetect it using the following steps:
 1. `Log4j`, if it's available,
 2. Then `JDK1.4` logging,

3. Then fallback to our STDERR logging.

- `DBMD.getTables()` shouldn't return tables if views are asked for, even if the database version doesn't support views. (Bug #9778)
- Fixed driver not returning `true` for `-1` when `ResultSet.getBoolean()` was called on result sets returned from server-side prepared statements.
- Added a `Manifest.MF` file with implementation information to the `.jar` file.
- More tests in `Field.isOpaqueBinary()` to distinguish opaque binary (that is, fields with type `CHAR(n)` and `CHARACTER SET BINARY`) from output of various scalar and aggregate functions that return strings.
- Should accept `null` for catalog (meaning use current) in DBMD methods, even though it's not JDBC-compliant for legacy's sake. Disable by setting connection property `nullCatalogMeansCurrent` to `false` (which will be the default value in C/J 3.2.x). (Bug #9917)
- Should accept `null` for name patterns in DBMD (meaning '%'), even though it isn't JDBC compliant, for legacy's sake. Disable by setting connection property `nullNamePatternMatchesAll` to `false` (which will be the default value in C/J 3.2.x). (Bug #9769)

D.5.11. Changes in MySQL Connector/J 3.1.7-stable (18 February 2005)

- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug #7686)
- Timestamps converted incorrectly to strings with server-side prepared statements and updatable result sets. (Bug #7715)
- Detect new `sql_mode` variable in string form (it used to be integer) and adjust quoting method for strings appropriately.
- Added `holdResultsOpenOverStatementClose` property (default is `false`), that keeps result sets open over `statement.close()` or new execution on same statement (suggested by Kevin Burton).

- Infinite recursion when “falling back” to master in failover configuration. (Bug #7952)
- Disable multi-statements (if enabled) for MySQL-4.1 versions prior to version 4.1.10 if the query cache is enabled, as the server returns wrong results in this configuration.
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly.
- Removed `dontUnpackBinaryResults` functionality, the driver now always stores results from server-side prepared statements as is from the server and unpacks them on demand.
- Emulated locators corrupt binary data when using server-side prepared statements. (Bug #8096)
- Fixed synchronization issue with `ServerPreparedStatement.serverPrepare()` that could cause deadlocks/crashes if connection was shared between threads.
- By default, the driver now scans SQL you are preparing via all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this by passing `emulateUnsupportedPstmts=false` in your JDBC URL. (Bug #4718)
- Remove `_binary` introducer from parameters used as in/out parameters in `CallableStatement`.
- Always return `byte[]`s for output parameters registered as `*BINARY`.
- Send correct value for “boolean” true to server for `PreparedStatement.setObject(n, "true", Types.BIT)`.
- Fixed bug with `Connection` not caching statements from `prepareStatement()` when the statement wasn't a server-side prepared statement.

- Choose correct “direction” to apply time adjustments when both client and server are in GMT time zone when using `ResultSet.get(..., cal)` and `PreparedStatement.set(..., cal)`.
- Added `dontTrackOpenResources` option (default is `false`, to be JDBC compliant), which helps with memory use for non-well-behaved apps (that is, applications that don't close `Statement` objects when they should).
- `ResultSet.getString()` doesn't maintain format stored on server, bug fix only enabled when `noDatetimeStringSync` property is set to `true` (the default is `false`). (Bug #8428)
- Fixed NPE in `ResultSet.realClose()` when using usage advisor and result set was already closed.
- `PreparedStatements` not creating streaming result sets. (Bug #8487)
- Don't pass `NULL` to `String.valueOf()` in `ResultSet.getNativeConvertToString()`, as it stringifies it (that is, returns `null`), which is not correct for the method in question.
- `ResultSet.getBigDecimal()` throws exception when rounding would need to occur to set scale. The driver now chooses a rounding mode of “half up” if non-rounding `BigDecimal.setScale()` fails. (Bug #8424)
- Added `useLocalSessionState` configuration property, when set to `true` the JDBC driver trusts that the application is well-behaved and only sets `autocommit` and `transaction isolation` levels using the methods provided on `java.sql.Connection`, and therefore can manipulate these values in many cases without incurring round-trips to the database server.
- Added `enableStreamingResults()` to `Statement` for connection pool implementations that check `Statement.setFetchSize()` for specification-compliant values. Call `Statement.setFetchSize(>=0)` to disable the streaming results for that statement.
- Added support for `BIT` type in MySQL-5.0.3. The driver will treat `BIT(1-8)` as the JDBC standard `BIT` type (which maps to `java.lang.Boolean`), as the server does not currently send enough information to determine the size of a bitfield when `< 9` bits are declared. `BIT(>9)` will be treated as `VARBINARY`,

and will return `byte[]` when `getObject()` is called.

D.5.12. Changes in MySQL Connector/J 3.1.6-stable (23 December 2004)

- Fixed hang on `SocketInputStream.read()` with `Statement.setMaxRows()` and multiple result sets when driver has to truncate result set directly, rather than tacking a `LIMIT n` on the end of it.
- `DBMD.getProcedures()` doesn't respect catalog parameter. (Bug #7026)

D.5.13. Changes in MySQL Connector/J 3.1.5-gamma (02 December 2004)

- Fix comparisons made between string constants and dynamic strings that are converted with either `toUpperCase()` or `toLowerCase()` to use `Locale.ENGLISH`, as some locales “override” case rules for English. Also use `StringUtils.indexOfIgnoreCase()` instead of `.toUpperCase().indexOf()`, avoids creating a very short-lived transient `String` instance.
- Server-side prepared statements did not honor `zeroDateTimeBehavior` property, and would cause class-cast exceptions when using `ResultSet.getObject()`, as the all-zero string was always returned. (Bug #5235)
- Fixed batched updates with server prepared statements weren't looking if the types had changed for a given batched set of parameters compared to the previous set, causing the server to return the error “Wrong arguments to `mysql_stmt_execute()`”.
- Handle case when string representation of timestamp contains trailing `‘.’` with no numbers following it.
- Inefficient detection of pre-existing string instances in `ResultSet.getNativeString()`. (Bug #5706)
- Don't throw exceptions for `Connection.releaseSavepoint()`.

- Use a per-session `Calendar` instance by default when decoding dates from `ServerPreparedStatement`s (set to `old`, less performant behavior by setting property `dynamicCalendars=true`).
- Added experimental configuration property `dontUnpackBinaryResults`, which delays unpacking binary result set values until they're asked for, and only creates object instances for non-numerical values (it is set to `false` by default). For some usecase/jvm combinations, this is friendlier on the garbage collector.
- `UNSIGNED BIGINT` unpacked incorrectly from server-side prepared statement result sets. (Bug #5729)
- `ServerSidePreparedStatement` allocating short-lived objects unnecessarily. (Bug #6225)
- Removed unwanted new `Throwable()` in `ResultSet` constructor due to bad merge (caused a new object instance that was never used for every result set created). Found while profiling for Bug #6359.
- Fixed too-early creation of `StringBuffer` in `EscapeProcessor.escapeSQL()`, also return `String` when escaping not needed (to avoid unnecessary object allocations). Found while profiling for Bug #6359.
- Use null-safe-equals for key comparisons in updatable result sets.
- `SUM()` on `DECIMAL` with server-side prepared statement ignores scale if zero-padding is needed (this ends up being due to conversion to `DOUBLE` by server, which when converted to a string to parse into `BigDecimal`, loses all “padding” zeros). (Bug #6537)
- Use `DatabaseMetaData.getIdentifierQuoteString()` when building DBMD queries.
- Use 1MB packet for sending file for `LOAD DATA LOCAL INFILE` if that is `< max_allowed_packet` on server.
- `ResultSetMetaData.getColumnDisplaySize()` returns incorrect values for multi-byte charsets. (Bug #6399)

- Make auto-deserialization of `java.lang.Objects` stored in BLOB columns configurable via `autoDeserialize` property (defaults to `false`).
- Re-work `Field.isOpaqueBinary()` to detect `CHAR(n) CHARACTER SET BINARY` to support fixed-length binary fields for `ResultSet.getObject()`.
- Use our own implementation of buffered input streams to get around blocking behavior of `java.io.BufferedReader`. Disable this with `useReadAheadInput=false`.
- Failing to connect to the server when one of the addresses for the given host name is IPV6 (which the server does not yet bind on). The driver now loops through *all* IP addresses for a given host, and stops on the first one that `accepts()` a `socket.connect()`. (Bug #6348)

D.5.14. Changes in MySQL Connector/J 3.1.4-beta (04 September 2004)

- Connector/J 3.1.3 beta does not handle integers correctly (caused by changes to support unsigned reads in `Buffer.readInt()` -> `Buffer.readShort()`). (Bug #4510)
- Added support in `DatabaseMetaData.getTables()` and `getTableTypes()` for views, which are now available in MySQL server 5.0.x.
- `ServerPreparedStatement.execute*()` sometimes threw `ArrayIndexOutOfBoundsException` when unpacking field metadata. (Bug #4642)
- Optimized integer number parsing, enable “old” slower integer parsing using JDK classes via `useFastIntParsing=false` property.
- Added `useOnlyServerErrorMessages` property, which causes message text in exceptions generated by the server to only contain the text sent by the server (as opposed to the `SQLState`'s “standard” description, followed by the server's error message). This property is set to `true` by default.
- `ResultSet.isNull()` does not work for primitives if a previous `null` was returned. (Bug #4689)

- Track packet sequence numbers if `enablePacketDebug=true`, and throw an exception if packets received out-of-order.
- `ResultSet.getObject()` returns wrong type for strings when using prepared statements. (Bug #4482)
- Calling `MysqlPooledConnection.close()` twice (even though an application error), caused NPE. Fixed.
- `ServerPreparedStatements` dealing with return of `DECIMAL` type don't work. (Bug #5012)
- `ResultSet.getObject()` doesn't return type `Boolean` for pseudo-bit types from prepared statements on 4.1.x (shortcut for avoiding extra type conversion when using binary-encoded result sets obscured test in `getObject()` for "pseudo" bit type). (Bug #5032)
- You can now use URLs in `LOAD DATA LOCAL INFILE` statements, and the driver will use Java's built-in handlers for retrieving the data and sending it to the server. This feature is not enabled by default, you must set the `allowUrlInLocalInfile` connection property to `true`.
- The driver is more strict about truncation of numerics on `ResultSet.get*()`, and will throw an `SQLException` when truncation is detected. You can disable this by setting `jdbcCompliantTruncation` to `false` (it is enabled by default, as this functionality is required for JDBC compliance).
- Added three ways to deal with all-zero datetimes when reading them from a `ResultSet`: `exception` (the default), which throws an `SQLException` with an `SQLState` of `S1009`; `convertToNull`, which returns `NULL` instead of the date; and `round`, which rounds the date to the nearest closest value which is `'0001-01-01'`.
- Fixed `ServerPreparedStatement` to read prepared statement metadata off the wire, even though it's currently a placeholder instead of using `MysqlIO.clearInputStream()` which didn't work at various times because data wasn't available to read from the server yet. This fixes sporadic errors users were having with `ServerPreparedStatements` throwing `ArrayIndexOutOfBoundsExceptions`.

- Use `com.mysql.jdbc.Message`'s classloader when loading resource bundle, should fix sporadic issues when the caller's classloader can't locate the resource bundle.

D.5.15. Changes in MySQL Connector/J 3.1.3-beta (07 July 2004)

- Mangle output parameter names for `CallableStatements` so they will not clash with user variable names.
- Added support for `INOUT` parameters in `CallableStatements`.
- Null bitmask sent for server-side prepared statements was incorrect. (Bug #4119)
- Use SQL Standard SQL states by default, unless `useSqlStateCodes` property is set to `false`.
- Added packet debugging code (see the `enablePacketDebug` property documentation).
- Added constants for MySQL error numbers (publicly accessible, see `com.mysql.jdbc.MySQLErrorNumbers`), and the ability to generate the mappings of vendor error codes to SQLStates that the driver uses (for documentation purposes).
- Externalized more messages (on-going effort).
- Error in retrieval of `mediumint` column with prepared statements and binary protocol. (Bug #4311)
- Support new time zone variables in MySQL-4.1.3 when `useTimezone=true`.
- Support for unsigned numerics as return types from prepared statements. This also causes a change in `ResultSet.getObject()` for the `bigint` unsigned type, which used to return `BigDecimal` instances, it now returns instances of `java.lang.BigInteger`.

D.5.16. Changes in MySQL Connector/J 3.1.2-alpha (09 June

2004)

- Fixed stored procedure parameter parsing info when size was specified for a parameter (for example, `char()`, `varchar()`).
- Enabled callable statement caching via `cacheCallableStmts` property.
- Fixed case when no output parameters specified for a stored procedure caused a bogus query to be issued to retrieve out parameters, leading to a syntax error from the server.
- Fixed case when no parameters could cause a `NullPointerException` in `CallableStatement.setOutputParameters()`.
- Removed wrapping of exceptions in `MysqlIO.changeUser()`.
- Fixed sending of split packets for large queries, enabled nio ability to send large packets as well.
- Added `.toString()` functionality to `ServerPreparedStatement`, which should help if you're trying to debug a query that is a prepared statement (it shows SQL as the server would process).
- Added `gatherPerformanceMetrics` property, along with properties to control when/where this info gets logged (see docs for more info).
- `ServerPreparedStatements` weren't actually de-allocating server-side resources when `.close()` was called.
- Added `logSlowQueries` property, along with `slowQueriesThresholdMillis` property to control when a query should be considered "slow."
- Correctly map output parameters to position given in `prepareCall()` versus. order implied during `registerOutParameter()`. (Bug #3146)
- Correctly detect initial character set for servers $\geq 4.1.0$.
- Cleaned up detection of server properties.

- Support placeholder for parameter metadata for server \geq 4.1.2.
- `getProcedures()` does not return any procedures in result set. (Bug #3539)
- `getProcedureColumns()` doesn't work with wildcards for procedure name. (Bug #3540)
- `DBMD.getSQLStateType()` returns incorrect value. (Bug #3520)
- Added `connectionCollation` property to cause driver to issue `set collation_connection=...` query on connection init if default collation for given charset is not appropriate.
- Fixed `DatabaseMetaData.getProcedures()` when run on MySQL-5.0.0 (output of `SHOW PROCEDURE STATUS` changed between 5.0.0 and 5.0.1).
- `getWarnings()` returns `SQLWarning` instead of `DataTruncation`. (Bug #3804)
- Don't enable server-side prepared statements for server version 5.0.0 or 5.0.1, as they aren't compatible with the '4.1.2+' style that the driver uses (the driver expects information to come back that isn't there, so it hangs).

D.5.17. Changes in MySQL Connector/J 3.1.1-alpha (14 February 2004)

- Fixed bug with `UpdatableResultSets` not using client-side prepared statements.
- Fixed character encoding issues when converting bytes to ASCII when MySQL doesn't provide the character set, and the JVM is set to a multi-byte encoding (usually affecting retrieval of numeric values).
- Unpack "unknown" data types from server prepared statements as `Strings`.
- Implemented long data (Blobs, Clobs, `InputStreams`, `Readers`) for server prepared statements.
- Implemented `Statement.getWarnings()` for MySQL-4.1 and newer (using `SHOW WARNINGS`).

- Default result set type changed to `TYPE_FORWARD_ONLY` (JDBC compliance).
- Centralized setting of result set type and concurrency.
- Refactored how connection properties are set and exposed as `DriverPropertyInfo` as well as `Connection` and `DataSource` properties.
- Support for NIO. Use `useNIO=true` on platforms that support NIO.
- Support for transaction savepoints (MySQL \geq 4.0.14 or 4.1.1).
- Support for `mysql_change_user()`. See the `changeUser()` method in `com.mysql.jdbc.Connection`.
- Reduced number of methods called in average query to be more efficient.
- Prepared statements will be re-prepared on auto-reconnect. Any errors encountered are postponed until first attempt to re-execute the re-prepared statement.
- Ensure that warnings are cleared before executing queries on prepared statements, as-per JDBC spec (now that we support warnings).
- Support “old” `profileSql` capitalization in `ConnectionProperties`. This property is deprecated, you should use `profileSQL` if possible.
- Optimized `Buffer.readLenByteArray()` to return shared empty byte array when length is 0.
- Allow contents of `PreparedStatement.setBlob()` to be retained between calls to `.execute*()`.
- Deal with 0-length tokens in `EscapeProcessor` (caused by callable statement escape syntax).
- Check for closed connection on delete/update/insert row operations in `UpdatableResultSet`.
- Fix support for table aliases when checking for all primary keys in `UpdatableResultSet`.

- Removed useFastDates connection property.
- Correctly initialize datasource properties from JNDI Refs, including explicitly specified URLs.
- DatabaseMetaData now reports supportsStoredProcedures() for MySQL versions >= 5.0.0
- Fixed stack overflow in Connection.prepareCall() (bad merge).
- Fixed IllegalArgumentException to Calendar.getTimeInMillis() in DateTimeValue (for JDK < 1.4).
- DatabaseMetaData.getColumns() is not returning correct column ordinal info for non-'%' column name patterns. (Bug #1673)
- Merged fix of datatype mapping from MySQL type FLOAT to java.sql.Types.REAL from 3.0 branch.
- Detect collation of column for RSMD.isCaseSensitive().
- Fixed sending of queries larger than 16M.
- Added named and indexed input/output parameter support to CallableStatement. MySQL-5.0.x or newer.
- Fixed NullPointerException in ServerPreparedStatement.setTimestamp(), as well as year and month discrepancies in ServerPreparedStatement.setTimestamp(), setDate().
- Added ability to have multiple database/JVM targets for compliance and regression/unit tests in build.xml.
- Fixed NPE and year/month bad conversions when accessing some datetime functionality in ServerPreparedStatements and their resultant result sets.
- Display where/why a connection was implicitly closed (to aid debugging).
- CommunicationsException implemented, that tries to determine why communications was lost with a server, and displays possible reasons when

.getMessage() is called.

- NULL values for numeric types in binary encoded result sets causing NullPointerExceptions. (Bug #2359)
- Implemented Connection.prepareCall(), and DatabaseMetaData.getProcedures() and getProcedureColumns().
- Reset long binary parameters in ServerPreparedStatement when clearParameters() is called, by sending COM_RESET_STMT to the server.
- Merged prepared statement caching, and .getMetaData() support from 3.0 branch.
- Fixed off-by-1900 error in some cases for years in TimeUtil.fastDate/TimeCreate() when unpacking results from server-side prepared statements.
- Fixed charset conversion issue in getTables(). (Bug #2502)
- Implemented multiple result sets returned from a statement or stored procedure.
- Server-side prepared statements were not returning datatype YEAR correctly. (Bug #2606)
- Enabled streaming of result sets from server-side prepared statements.
- Class-cast exception when using scrolling result sets and server-side prepared statements. (Bug #2623)
- Merged unbuffered input code from 3.0.
- Fixed ConnectionProperties that weren't properly exposed via accessors, cleaned up ConnectionProperties code.
- NULL fields were not being encoded correctly in all cases in server-side prepared statements. (Bug #2671)
- Fixed rare buffer underflow when writing numbers into buffers for sending

prepared statement execution requests.

- Use DocBook version of docs for shipped versions of drivers.

D.5.18. Changes in MySQL Connector/J 3.1.0-alpha (18 February 2003)

- Added `requiresSSL` property.
- Added `useServerPrepStmts` property (default `false`). The driver will use server-side prepared statements when the server version supports them (4.1 and newer) when this property is set to `true`. It is currently set to `false` by default until all bind/fetch functionality has been implemented. Currently only DML prepared statements are implemented for 4.1 server-side prepared statements.
- Track open Statements, close all when `Connection.close()` is called (JDBC compliance).

D.5.19. Changes in MySQL Connector/J 3.0.17-ga (23 June 2005)

- Timestamp/Time conversion goes in the wrong “direction” when `useTimeZone=true` and server time zone differs from client time zone. (Bug #5874)
- `DatabaseMetaData.getIndexInfo()` ignored unique parameter. (Bug #7081)
- Support new protocol type `MYSQL_TYPE_VARCHAR`.
- Added `useOldUTF8Behavior` configuration property, which causes JDBC driver to act like it did with MySQL-4.0.x and earlier when the character encoding is `utf-8` when connected to MySQL-4.1 or newer.
- Statements created from a pooled connection were returning physical connection instead of logical connection when `getConnection()` was called. (Bug #7316)
- PreparedStatements don't encode Big5 (and other multi-byte) character

sets correctly in static SQL strings. (Bug #7033)

- Connections starting up failed-over (due to down master) never retry master. (Bug #6966)
- `PreparedStatement.fixDecimalExponent()` adding extra +, making number unparseable by MySQL server. (Bug #7061)
- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug #7686)
- Backported SQLState codes mapping from Connector/J 3.1, enable with `useSqlStateCodes=true` as a connection property, it defaults to `false` in this release, so that we don't break legacy applications (it defaults to `true` starting with Connector/J 3.1).
- `PreparedStatement.fixDecimalExponent()` adding extra +, making number unparseable by MySQL server. (Bug #7601)
- Escape sequence `{fn convert(..., type)}` now supports ODBC-style types that are prepended by `SQL_`.
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly.
- Handle streaming result sets with more than 2 billion rows properly by fixing wraparound of row number counter.
- MS932, SHIFT_JIS, and Windows_31J not recognized as aliases for sjis. (Bug #7607)
- Adding CP943 to aliases for sjis. (Bug #6549, fixed while fixing Bug #7607)
- Which requires hex escaping of binary data when using multi-byte charsets with prepared statements. (Bug #8064)
- `NON_UNIQUE` column from `DBMD.getIndexInfo()` returned inverted value. (Bug #8812)

- Workaround for server Bug #9098: Default values of CURRENT_* for DATE, TIME, DATETIME, and TIMESTAMP columns can't be distinguished from string values, so `UpdatableResultSet.moveToInsertRow()` generates bad SQL for inserting default values.
- EUCKR charset is sent as SET NAMES euc_kr which MySQL-4.1 and newer doesn't understand. (Bug #8629)
- `DatabaseMetaData.supportsSelectForUpdate()` returns correct value based on server version.
- Use hex escapes for `PreparedStatement.setBytes()` for double-byte charsets including “aliases” windows-31J, CP934, MS932.
- Added support for the EUC_JP_Solaris character encoding, which maps to a MySQL encoding of eucjpms (backported from 3.1 branch). This only works on servers that support eucjpms, namely 5.0.3 or later.

D.5.20. Changes in MySQL Connector/J 3.0.16-ga (15 November 2004)

- Re-issue character set configuration commands when re-using pooled connections and/or `Connection.changeUser()` when connected to MySQL-4.1 or newer.
- Fixed `ResultSetMetaData.isReadOnly()` to detect non-writable columns when connected to MySQL-4.1 or newer, based on existence of “original” table and column names.
- `ResultSet.updateByte()` when on insert row throws `ArrayOutOfBoundsException`. (Bug #5664)
- Fixed `DatabaseMetaData.getTypes()` returning incorrect (this is, non-negative) scale for the NUMERIC type.
- Off-by-one bug in `Buffer.readString(string)`. (Bug #5664)
- Made `TINYINT(1)` -> BIT/Boolean conversion configurable via `tinyInt1isBit` property (default true to be JDBC compliant out of the box).

- Only set `character_set_results` during connection establishment if server version \geq 4.1.1.
- Fixed regression where `useUnbufferedInput` was defaulting to `false`.
- `ResultSet.getTimestamp()` on a column with `TIME` in it fails. (Bug #5664)

D.5.21. Changes in MySQL Connector/J 3.0.15-production (04 September 2004)

- `StringUtils.escapeEasternUnicodeByteStream` was still broken for GBK. (Bug #4010)
- Failover for `autoReconnect` not using port numbers for any hosts, and not retrying all hosts. (**Warning:** This required a change to the `SocketFactory.connect()` method signature, which is now `public Socket connect(String host, int portNumber, Properties props)`; therefore, any third-party socket factories will have to be changed to support this signature. (Bug #4334)
- Logical connections created by `MySQLConnectionPoolDataSource` will now issue a `rollback()` when they are closed and sent back to the pool. If your application server/connection pool already does this for you, you can set the `rollbackOnPooledClose` property to `false` to avoid the overhead of an extra `rollback()`.
- Removed redundant calls to `checkRowPos()` in `ResultSet`.
- `DOUBLE` mapped twice in `DBMD.getTypeInfo()`. (Bug #4742)
- Added FLOSS license exemption.
- Calling `.close()` twice on a `PooledConnection` causes NPE. (Bug #4808)
- `DBMD.getColumns()` returns incorrect JDBC type for unsigned columns. This affects type mappings for all numeric types in the `RSMD.getColumnType()` and `RSMD.getColumnTypeNames()` methods as well, to ensure that “like” types from `DBMD.getColumns()` match up with what `RSMD.getColumnType()` and `getColumnTypeNames()` return. (Bug #4138,

Bug #4860)

- “Production” is now “GA” (General Availability) in naming scheme of distributions.
- `RSMD.getPrecision()` returning 0 for non-numeric types (should return max length in chars for non-binary types, max length in bytes for binary types). This fix also fixes mapping of `RSMD.getColumnType()` and `RSMD.getColumnTypeName()` for the BLOB types based on the length sent from the server (the server doesn't distinguish between TINYBLOB, BLOB, MEDIUMBLOB or LONGBLOB at the network protocol level). (Bug #4880)
- `ResultSet` should release `Field[]` instance in `.close()`. (Bug #5022)
- `ResultSet.getMetaData()` should not return incorrectly initialized metadata if the result set has been closed, but should instead throw an `SQLException`. Also fixed for `getRow()` and `getWarnings()` and traversal methods by calling `checkClosed()` before operating on instance-level fields that are nullified during `.close()`. (Bug #5069)
- Parse new time zone variables from 4.1.x servers.
- Use `_binary` introducer for `PreparedStatement.setBytes()` and `set*Stream()` when connected to MySQL-4.1.x or newer to avoid misinterpretation during character conversion.

D.5.22. Changes in MySQL Connector/J 3.0.14-production (28 May 2004)

- Fixed URL parsing error.

D.5.23. Changes in MySQL Connector/J 3.0.13-production (27 May 2004)

- Using a `MySQLDataSource` without server name fails. (Bug #3848)
- No Database Selected when using `MysqlConnectionPoolDataSource`. (Bug #3920)

- `PreparedStatement.getGeneratedKeys()` method returns only 1 result for batched insertions. (Bug #3873)

D.5.24. Changes in MySQL Connector/J 3.0.12-production (18 May 2004)

- Add unsigned attribute to `DatabaseMetaData.getColumns()` output in the `TYPE_NAME` column.
- Added `failOverReadOnly` property, to allow end-user to configure state of connection (read-only/writable) when failed over.
- Backported “change user” and “reset server state” functionality from 3.1 branch, to allow clients of `MysqlConnectionPoolDataSource` to reset server state on `getConnection()` on a pooled connection.
- Don't escape SJIS/GBK/BIG5 when using MySQL-4.1 or newer.
- Allow `url` parameter for `MysqlDataSource` and `MysqlConnectionPoolDataSource` so that passing of other properties is possible from inside appservers.
- Map duplicate key and foreign key errors to `SQLState` of 23000.
- Backport documentation tooling from 3.1 branch.
- Return creating statement for `ResultSets` created by `getGeneratedKeys()`. (Bug #2957)
- Allow `java.util.Date` to be sent in as parameter to `PreparedStatement.setObject()`, converting it to a `Timestamp` to maintain full precision. (Bug #103).
- Don't truncate BLOB or CLOB values when using `setBytes()` and/or `setBinary/CharacterStream()`. (Bug #2670).
- Dynamically configure character set mappings for field-level character sets on MySQL-4.1.0 and newer using `SHOW COLLATION` when connecting.
- Map binary character set to US-ASCII to support DATETIME charset

recognition for servers $\geq 4.1.2$.

- Use `SET character_set_results` during initialization to allow any charset to be returned to the driver for result sets.
- Use `charsetnr` returned during connect to encode queries before issuing `SET NAMES` on MySQL $\geq 4.1.0$.
- Add helper methods to `ResultSetMetaData` (`getColumnCharacterEncoding()` and `getColumnCharacterSet()`) to allow end-users to see what charset the driver thinks it should be using for the column.
- Only set `character_set_results` for MySQL $\geq 4.1.0$.
- `StringUtils.escapeSJISByteStream()` not covering all eastern double-byte charsets correctly. (Bug #3511)
- Renamed `StringUtils.escapeSJISByteStream()` to more appropriate `escapeEasternUnicodeByteStream()`.
- Not specifying database in URL caused `MalformedURLException` exception. (Bug #3554)
- Auto-convert MySQL encoding names to Java encoding names if used for `characterEncoding` property.
- Added encoding names that are recognized on some JVMs to fix case where they were reverse-mapped to MySQL encoding names incorrectly.
- Use `junit.textui.TestRunner` for all unit tests (to allow them to be run from the command line outside of Ant or Eclipse).
- `UpdatableResultSet` not picking up default values for `moveToInsertRow()`. (Bug #3557)
- Inconsistent reporting of data type. The server still doesn't return all types for `*BLOBs` `*TEXT` correctly, so the driver won't return those correctly. (Bug #3570)

- `DBMD.getSQLStateType()` returns incorrect value. (Bug #3520)
- Fixed regression in `PreparedStatement.setString()` and eastern character encodings.
- Made `StringRegressionTest 4.1-unicode` aware.

D.5.25. Changes in MySQL Connector/J 3.0.11-stable (19 February 2004)

- Trigger a `SET NAMES utf8` when encoding is forced to `utf8` or `utf-8` via the `characterEncoding` property. Previously, only the Java-style encoding name of `utf-8` would trigger this.
- `AutoReconnect` time was growing faster than exponentially. (Bug #2447)
- Fixed failover always going to last host in list. (Bug #2578)
- Added `useUnbufferedInput` parameter, and now use it by default (due to JVM issue <http://developer.java.sun.com/developer/bugParade/bugs/4401235.html>)
- Detect on/off or 1, 2, 3 form of `lower_case_table_names` value on server.
- Return `java.lang.Integer` for `TINYINT` and `SMALLINT` types from `ResultSetMetaData.getColumnClassName()`. (Bug #2852)
- Return `java.lang.Double` for `FLOAT` type from `ResultSetMetaData.getColumnClassName()`. (Bug #2855)
- Return `[B` instead of `java.lang.Object` for `BINARY`, `VARBINARY` and `LONGVARBINARY` types from `ResultSetMetaData.getColumnClassName()` (JDBC compliance).
- Issue connection events on all instances created from a `ConnectionPoolDataSource`.

D.5.26. Changes in MySQL Connector/J 3.0.10-stable (13 January 2004)

- Don't count quoted IDs when inside a 'string' in PreparedStatement parsing. (Bug #1511)
- “Friendlier” exception message for PacketTooLargeException. (Bug #1534)
- Backported fix for aliased tables and UpdatableResultSets in checkUpdatability() method from 3.1 branch.
- Fix for ArrayIndexOutOfBoundsException exception when using Statement.setMaxRows(). (Bug #1695)
- Barge blobs and split packets not being read correctly. (Bug #1576)
- Fixed regression of Statement.getGeneratedKeys() and REPLACE statements.
- Subsequent call to ResultSet.updateFoo() causes NPE if result set is not updatable. (Bug #1630)
- Fix for 4.1.1-style authentication with no password.
- Foreign Keys column sequence is not consistent in DatabaseMetaData.getImported/Exported/CrossReference(). (Bug #1731)
- DatabaseMetaData.getSystemFunction() returning bad function VResultsSion. (Bug #1775)
- Cross-database updatable result sets are not checked for updatability correctly. (Bug #1592)
- DatabaseMetaData.getColumns() should return Types.LONGVARCHAR for MySQL LONGTEXT type.
- ResultSet.getObject() on TINYINT and SMALLINT columns should return Java type Integer. (Bug #1913)
- Added alwaysClearStream connection property, which causes the driver to always empty any remaining data on the input stream before each query.

- Added more descriptive error message `Server Configuration Denies Access to DataSource`, as well as retrieval of message from server.
- Autoreconnect code didn't set catalog upon reconnect if it had been changed.
- Implement `ResultSet.updateClob()`.
- `ResultSetMetaData.isCaseSensitive()` returned wrong value for CHAR/VARCHAR columns.
- Connection property `maxRows` not honored. (Bug #1933)
- Statements being created too many times in `DBMD.extractForeignKeyFromCreateTable()`. (Bug #1925)
- Support escape sequence `{fn convert ... }`. (Bug #1914)
- `ArrayIndexOutOfBoundsException` when parameter number == number of parameters + 1. (Bug #1958)
- `ResultSet.findColumn()` should use first matching column name when there are duplicate column names in SELECT query (JDBC-compliance). (Bug #2006)
- Removed static synchronization bottleneck from `PreparedStatement.setTimestamp()`.
- Removed static synchronization bottleneck from instance factory method of `SingleByteCharsetConverter`.
- Enable caching of the parsing stage of prepared statements via the `cachePrepStmts`, `prepStmtCacheSize`, and `prepStmtCacheSqlLimit` properties (disabled by default).
- Speed up parsing of `PreparedStatements`, try to use one-pass whenever possible.
- Fixed security exception when used in Applets (applets can't read the system property `file.encoding` which is needed for `LOAD DATA LOCAL`

INFILE).

- Use constants for SQLStates.
- Map charset ko18_ru to ko18r when connected to MySQL-4.1.0 or newer.
- Ensure that `Buffer.writeString()` saves room for the `\0`.
- Fixed exception `Unknown character set 'danish'` on connect with JDK-1.4.0
- Fixed mappings in `SQLException` to report deadlocks with SQLStates of 41000.
- `maxRows` property would affect internal statements, so check it for all statement creation internal to the driver, and set to 0 when it is not.

D.5.27. Changes in MySQL Connector/J 3.0.9-stable (07 October 2003)

- Faster date handling code in `ResultSet` and `PreparedStatement` (no longer uses `Date` methods that synchronize on static calendars).
- Fixed test for end of buffer in `Buffer.readString()`.
- Fixed `ResultSet.previous()` behavior to move current position to before result set when on first row of result set. (Bug #496)
- Fixed `Statement` and `PreparedStatement` issuing bogus queries when `setMaxRows()` had been used and a `LIMIT` clause was present in the query.
- `refreshRow` didn't work when primary key values contained values that needed to be escaped (they ended up being doubly escaped). (Bug #661)
- Support InnoDB constraint names when extracting foreign key information in `DatabaseMetaData` (implementing ideas from Parwinder Sekhon). (Bug #517, Bug #664)
- Backported 4.1 protocol changes from 3.1 branch (server-side SQL states, new field information, larger client capability flags, connect-with-database, and so forth).

- Fix `UpdatableResultSet` to return values for `getXXX()` when on insert row. (Bug #675)
- The `insertRow` in an `UpdatableResultSet` is now loaded with the default column values when `moveToInsertRow()` is called. (Bug #688)
- `DatabaseMetaData.getColumns()` wasn't returning `NULL` for default values that are specified as `NULL`.
- Change default statement type/concurrency to `TYPE_FORWARD_ONLY` and `CONCUR_READ_ONLY` (spec compliance).
- Don't try and reset isolation level on reconnect if MySQL doesn't support them.
- Don't wrap `SQLExceptions` in `RowDataDynamic`.
- Don't change timestamp TZ twice if `useTimezone==true`. (Bug #774)
- Fixed regression in large split-packet handling. (Bug #848)
- Better diagnostic error messages in exceptions for “streaming” result sets.
- Issue exception on `ResultSet.getXXX()` on empty result set (wasn't caught in some cases).
- Don't hide messages from exceptions thrown in I/O layers.
- Don't fire connection closed events when closing pooled connections, or on `PooledConnection.getConnection()` with already open connections. (Bug #884)
- Clip +/- INF (to smallest and largest representative values for the type in MySQL) and NaN (to 0) for `setDouble/setFloat()`, and issue a warning on the statement when the server does not support +/- INF or NaN.
- Double-escaping of `'\'` when charset is SJIS or GBK and `'\'` appears in non-escaped input. (Bug #879)
- When emptying input stream of unused rows for “streaming” result sets,

have the current thread `yield()` every 100 rows in order to not monopolize CPU time.

- `DatabaseMetaData.getColumns()` getting confused about the keyword “set” in character columns. (Bug #1099)
- Fixed deadlock issue with `Statement.setMaxRows()`.
- Fixed `CLOB.truncate()`. (Bug #1130)
- Optimized `CLOB.setCharacterStream()`. (Bug #1131)
- Made `databaseName`, `portNumber`, and `serverName` optional parameters for `MysqlDataSourceFactory`. (Bug #1246)
- `ResultSet.get/setString` mashing char 127. (Bug #1247)
- Backported authentication changes for 4.1.1 and newer from 3.1 branch.
- Added `com.mysql.jdbc.util.BaseBugReport` to help creation of testcases for bug reports.
- Added property to “clobber” streaming results, by setting the `clobberStreamingResults` property to `true` (the default is `false`). This will cause a “streaming” `ResultSet` to be automatically closed, and any outstanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server.

D.5.28. Changes in MySQL Connector/J 3.0.8-stable (23 May 2003)

- Allow bogus URLs in `Driver.getPropertyInfo()`.
- Return list of generated keys when using multi-value `INSERTS` with `Statement.getGeneratedKeys()`.
- Use JVM charset with filenames and `LOAD DATA [LOCAL] INFILE`.
- Fix infinite loop with `Connection.cleanup()`.

- Changed Ant target `compile-core` to `compile-driver`, and made test suite compilation a separate target.
- Fixed result set not getting set for `Statement.executeUpdate()`, which affected `getGeneratedKeys()` and `getUpdateCount()` in some cases.
- Unicode character `0xFFFF` in a string would cause the driver to throw an `ArrayOutOfBoundsException`. (Bug #378).
- Return correct number of generated keys when using `REPLACE` statements.
- Fix problem detecting server character set in some cases.
- Fix row data decoding error when using very large packets.
- Optimized row data decoding.
- Issue exception when operating on an already closed prepared statement.
- Fixed SJIS encoding bug, thanks to Naoto Sato.
- Optimized usage of `EscapeProcessor`.
- Allow multiple calls to `Statement.close()`.

D.5.29. Changes in MySQL Connector/J 3.0.7-stable (08 April 2003)

- Fixed `MysqlPooledConnection.close()` calling wrong event type.
- Fixed `StringIndexOutOfBoundsException` in `PreparedStatement.setClob()`.
- 4.1 Column Metadata fixes.
- Remove synchronization from `Driver.connect()` and `Driver.acceptsUrl()`.
- `IOExceptions` during a transaction now cause the `Connection` to be closed.

- Fixed missing conversion for YEAR type in `ResultSetMetaData.getColumnTypeName()`.
- Don't pick up indexes that start with `pri` as primary keys for `DBMD.getPrimaryKeys()`.
- Throw `SQLExceptions` when trying to do operations on a forcefully closed `Connection` (that is, when a communication link failure occurs).
- You can now toggle profiling on/off using `Connection.setProfileSql(boolean)`.
- Fixed charset issues with database metadata (charset was not getting set correctly).
- Updatable `ResultSets` can now be created for aliased tables/columns when connected to MySQL-4.1 or newer.
- Fixed `LOAD DATA LOCAL INFILE` bug when `file > max_allowed_packet`.
- Fixed escaping of `0x5c ('\\')` character for GBK and Big5 charsets.
- Fixed `ResultSet.getTimestamp()` when underlying field is of type `DATE`.
- Ensure that packet size from `alignPacketSize()` does not exceed `max_allowed_packet` (JVM bug)
- Don't reset `Connection.isReadOnly()` when `autoReconnecting`.

D.5.30. Changes in MySQL Connector/J 3.0.6-stable (18 February 2003)

- Fixed `ResultSetMetaData` to return "" when catalog not known. Fixes `NullPointerExceptions` with Sun's `CachedRowSet`.
- Fixed `DBMD.getTypeInfo()` and `DBMD.getColumns()` returning different value for precision in `TEXT` and `BLOB` types.
- Allow ignoring of warning for “non transactional tables” during rollback (compliance/usability) by setting `ignoreNonTxTables` property to `true`.

- Fixed `SQLExceptions` getting swallowed on initial connect.
- Fixed `Statement.setMaxRows()` to stop sending `LIMIT` type queries when not needed (performance).
- Clean up `Statement` query/method mismatch tests (that is, `INSERT` not allowed with `.executeQuery()`).
- More checks added in `ResultSet` traversal method to catch when in closed state.
- Fixed `ResultSetMetaData.isWritable()` to return correct value.
- Add “window” of different `NULL` sorting behavior to `DBMD.nullsAreSortedAtStart` (4.0.2 to 4.0.10, true; otherwise, no).
- Implemented `Blob.setBytes()`. You still need to pass the resultant `Blob` back into an updatable `ResultSet` or `PreparedStatement` to persist the changes, because `MySQL` does not support “locators”.
- Backported 4.1 charset field info changes from `Connector/J 3.1`.

D.5.31. Changes in MySQL Connector/J 3.0.5-gamma (22 January 2003)

- Fixed `Buffer.fastSkipLenString()` causing `ArrayIndexOutOfBoundsException` exceptions with some queries when unpacking fields.
- Implemented an empty `TypeMap` for `Connection.getTypeMap()` so that some third-party apps work with `MySQL` (`IBM WebSphere 5.0 Connection pool`).
- Added missing `LONGTEXT` type to `DBMD.getColumns()`.
- Retrieve `TX_ISOLATION` from database for `Connection.getTransactionIsolation()` when the `MySQL` version supports it, instead of an instance variable.
- Quote table names in `DatabaseMetaData.getColumns()`, `getPrimaryKeys()`, `getIndexInfo()`, `getBestRowIdentifier()`.

- Greatly reduce memory required for `setBinaryStream()` in `PreparedStatement`s.
- Fixed `ResultSet.isBeforeFirst()` for empty result sets.
- Added update options for foreign key metadata.

D.5.32. Changes in MySQL Connector/J 3.0.4-gamma (06 January 2003)

- Added quoted identifiers to database names for `Connection.setCatalog`.
- Added support for quoted identifiers in `PreparedStatement` parser.
- Streamlined character conversion and `byte[]` handling in `PreparedStatement`s for `setByte()`.
- Reduce memory footprint of `PreparedStatement`s by sharing outbound packet with `MySQLIO`.
- Added `strictUpdates` property to allow control of amount of checking for “correctness” of updatable result sets. Set this to `false` if you want faster updatable result sets and you know that you create them from `SELECT` statements on tables with primary keys and that you have selected all primary keys in your query.
- Added support for 4.0.8-style large packets.
- Fixed `PreparedStatement.executeBatch()` parameter overwriting.

D.5.33. Changes in MySQL Connector/J 3.0.3-dev (17 December 2002)

- Changed `charsToByte` in `SingleByteCharConverter` to be non-static.
- Changed `SingleByteCharConverter` to use lazy initialization of each converter.
- Fixed charset handling in `Fields.java`.

- Implemented `Connection.nativeSQL()`.
- More robust escape tokenizer: Recognize `--` comments, and allow nested escape sequences (see `testsuite.EscapeProcessingTest`).
- `DBMD.getImported/ExportedKeys()` now handles multiple foreign keys per table.
- Fixed `ResultSetMetaData.getPrecision()` returning incorrect values for some floating-point types.
- Fixed `ResultSetMetaData.getColumnTypeName()` returning BLOB for TEXT and TEXT for BLOB types.
- Fixed `Buffer.isLastDataPacket()` for 4.1 and newer servers.
- Added `CLIENT_LONG_FLAG` to be able to get more column flags (`isAutoIncrement()` being the most important).
- Because of above, implemented `ResultSetMetaData.isAutoIncrement()` to use `Field.isAutoIncrement()`.
- Honor `lower_case_table_names` when enabled in the server when doing table name comparisons in `DatabaseMetaData` methods.
- Some MySQL-4.1 protocol support (extended field info from selects).
- Use non-aliased table/column names and database names to fully qualify tables and columns in `UpdatableResultSet` (requires MySQL-4.1 or newer).
- Allow user to alter behavior of `Statement/PreparedStatement.executeBatch()` via `continueBatchOnError` property (defaults to true).
- Check for connection closed in more `Connection` methods (`createStatement`, `prepareStatement`, `setTransactionIsolation`, `setAutoCommit`).
- More robust implementation of updatable result sets. Checks that *all*

primary keys of the table have been selected.

- `LOAD DATA LOCAL INFILE . . .` now works, if your server is configured to allow it. Can be turned off with the `allowLoadLocalInfile` property (see the README).
- Substitute '?' for unknown character conversions in single-byte character sets instead of '\0'.
- `NamedPipeSocketFactory` now works (only intended for Windows), see README for instructions.

D.5.34. Changes in MySQL Connector/J 3.0.2-dev (08 November 2002)

- Fixed issue with updatable result sets and `PreparedStatement`s not working.
- Fixed `ResultSet.setFetchDirection(FETCH_UNKNOWN)`.
- Fixed issue when calling `Statement.setFetchSize()` when using arbitrary values.
- Fixed incorrect conversion in `ResultSet.getLong()`.
- Implemented `ResultSet.updateBlob()`.
- Removed duplicate code from `UpdatableResultSet` (it can be inherited from `ResultSet`, the extra code for each method to handle updatability I thought might someday be necessary has not been needed).
- Fixed `UnsupportedEncodingException` thrown when “forcing” a character encoding via properties.
- Fixed various non-ASCII character encoding issues.
- Added driver property `useHostsInPrivileges`. Defaults to true. Affects whether or not `@hostname` will be used in `DBMD.getColumn/TablePrivileges`.

- All DBMD result set columns describing schemas now return NULL to be more compliant with the behavior of other JDBC drivers for other database systems (MySQL does not support schemas).
- Added SSL support. See README for information on how to use it.
- Properly restore connection properties when autoReconnecting or failing-over, including autoCommit state, and isolation level.
- Use SHOW CREATE TABLE when possible for determining foreign key information for DatabaseMetaData. Also allows cascade options for DELETE information to be returned.
- Escape 0x5c character in strings for the SJIS charset.
- Fixed start position off-by-1 error in Clob.getSubString().
- Implemented Clob.truncate().
- Implemented Clob.setString().
- Implemented Clob.setAsciiStream().
- Implemented Clob.setCharacterStream().
- Added com.mysql.jdbc.MinAdmin class, which allows you to send shutdown command to MySQL server. This is intended to be used when “embedding” Java and MySQL server together in an end-user application.
- Added connectTimeout parameter that allows users of JDK-1.4 and newer to specify a maximum time to wait to establish a connection.
- Failover and autoReconnect work only when the connection is in an autoCommit(false) state, in order to stay transaction-safe.
- Added queriesBeforeRetryMaster property that specifies how many queries to issue when failed over before attempting to reconnect to the master (defaults to 50).
- Fixed DBMD.supportsResultSetConcurrency() so that it returns true for

`ResultSet.TYPE_SCROLL_INSENSITIVE` and `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`.

- Fixed `ResultSet.isLast()` for empty result sets (should return false).
- `PreparedStatement` now honors stream lengths in `setBinary/Ascii/Character Stream()` unless you set the connection property `useStreamLengthsInPrepStmts` to false.
- Removed some not-needed temporary object creation by smarter use of Strings in `EscapeProcessor`, `Connection` and `DatabaseMetaData` classes.

D.5.35. Changes in MySQL Connector/J 3.0.1-dev (21 September 2002)

- Fixed `ResultSet.getRow()` off-by-one bug.
- Fixed `RowDataStatic.getAt()` off-by-one bug.
- Added limited Clob functionality (`ResultSet.getClob()`, `PreparedStatement.setClob()`, `PreparedStatement.setObject(Clob)`).
- Added `socketTimeout` parameter to URL.
- `Connection.isClosed()` no longer “pings” the server.
- `Connection.close()` issues `rollback()` when `getAutoCommit()` is false.
- Added `paranoid` parameter, which sanitizes error messages by removing “sensitive” information from them (such as hostnames, ports, or usernames), as well as clearing “sensitive” data structures when possible.
- Fixed `ResultSetMetaData.isSigned()` for `TINYINT` and `BIGINT`.
- Charsets now automatically detected. Optimized code for single-byte character set conversion.
- Implemented `ResultSet.getCharacterStream()`.
- Added `LOCAL TEMPORARY` to table types in

`DatabaseMetaData.getTableTypes()`.

- Massive code clean-up to follow Java coding conventions (the time had come).

D.5.36. Changes in MySQL Connector/J 3.0.0-dev (31 July 2002)

- **!!! LICENSE CHANGE !!!** The driver is now GPL. If you need non-GPL licenses, please contact me <mark@mysql.com>.
- JDBC-3.0 functionality including `Statement/PreparedStatement.getGeneratedKeys()` and `ResultSet.getURL()`.
- Performance enhancements: Driver is now 50–100% faster in most situations, and creates fewer temporary objects.
- Repackaging: New driver name is `com.mysql.jdbc.Driver`, old name still works, though (the driver is now provided by MySQL-AB).
- Better checking for closed connections in `Statement` and `PreparedStatement`.
- Support for streaming (row-by-row) result sets (see README) Thanks to Doron.
- Support for large packets (new addition to MySQL-4.0 protocol), see README for more information.
- JDBC Compliance: Passes all tests besides stored procedure tests.
- Fix and sort primary key names in `DBMetaData` (SF bugs 582086 and 582086).
- Float types now reported as `java.sql.Types.FLOAT` (SF bug 579573).
- `ResultSet.getTimestamp()` now works for DATE types (SF bug 559134).
- `ResultSet.getDate/Time/Timestamp` now recognizes all forms of invalid values that have been set to all zeros by MySQL (SF bug 586058).

- Testsuite now uses Junit (which you can get from <http://www.junit.org>).
- The driver now only works with JDK-1.2 or newer.
- Added multi-host failover support (see README).
- General source-code cleanup.
- Overall speed improvements via controlling transient object creation in `MysqlIO` class when reading packets.
- Performance improvements in string handling and field metadata creation (lazily instantiated) contributed by Alex Twisleton-Wykeham-Fiennes.

D.5.37. Changes in MySQL Connector/J 2.0.14 (16 May 2002)

- More code cleanup.
- `PreparedStatement` now releases resources on `.close()`. (SF bug 553268)
- Quoted identifiers not used if server version does not support them. Also, if server started with `--ansi` or `--sql-mode=ANSI_QUOTES`, `"` will be used as an identifier quote character, otherwise `'` will be used.
- `ResultSet.getDouble()` now uses code built into JDK to be more precise (but slower).
- `LogicalHandle.isClosed()` calls through to physical connection.
- Added SQL profiling (to `STDERR`). Set `profileSql=true` in your JDBC URL. See README for more information.
- Fixed typo for `relaxAutoCommit` parameter.

D.5.38. Changes in MySQL Connector/J 2.0.13 (24 April 2002)

- More code cleanup.
- Fixed unicode chars being read incorrectly. (SF bug 541088)

- Faster blob escaping for PrepStmt.
- Added set/getPortNumber() to DataSource(s). (SF bug 548167)
- Added setURL() to MySQLXADataSource. (SF bug 546019)
- PreparedStatement.toString() fixed. (SF bug 534026)
- ResultSetMetaData.getColumnClassName() now implemented.
- Rudimentary version of Statement.getGeneratedKeys() from JDBC-3.0 now implemented (you need to be using JDK-1.4 for this to work, I believe).
- DBMetaData.getIndexInfo() - bad PAGES fixed. (SF BUG 542201)

D.5.39. Changes in MySQL Connector/J 2.0.12 (07 April 2002)

- General code cleanup.
- Added getIdleFor() method to Connection and MySQLLogicalHandle.
- Relaxed synchronization in all classes, should fix 520615 and 520393.
- Added getTable/ColumnPrivileges() to DBMD (fixes 484502).
- Added new types to getTypeInfo(), fixed existing types thanks to Al Davis and Kid Kalanon.
- Added support for BIT types (51870) to PreparedStatement.
- Fixed getRow() bug (527165) in ResultSet.
- Fixes for ResultSet updatability in PreparedStatement.
- Fixed time zone off-by-1-hour bug in PreparedStatement (538286, 528785).
- ResultSet: Fixed updatability (values being set to null if not updated).
- DataSources - fixed setUrl bug (511614, 525565), wrong datasource class

name (532816, 528767).

- Added identifier quoting to all DatabaseMetaData methods that need them (should fix 518108).
- Added support for YEAR type (533556).
- `ResultSet.insertRow()` should now detect `auto_increment` fields in most cases and use that value in the new row. This detection will not work in multi-valued keys, however, due to the fact that the MySQL protocol does not return this information.
- `ResultSet.refreshRow()` implemented.
- Fixed `testsuite.Traversal` after `Last()` bug, thanks to Igor Lastric.

D.5.40. Changes in MySQL Connector/J 2.0.11 (27 January 2002)

- Fixed missing `DELETE_RULE` value in `DBMD.getImported/ExportedKeys()` and `getCrossReference()`.
- Full synchronization of `Statement.java`.
- More changes to fix Unexpected end of input stream errors when reading BLOB values. This should be the last fix.

D.5.41. Changes in MySQL Connector/J 2.0.10 (24 January 2002)

- Fixed spurious Unexpected end of input stream errors in `MysqlIO` (bug 507456).
- Fixed null-pointer-exceptions when using `MysqlConnectionPoolDataSource` with Websphere 4 (bug 505839).

D.5.42. Changes in MySQL Connector/J 2.0.9 (13 January 2002)

- **Ant** build was corrupting included jar files, fixed (bug 487669).
- Fixed extra memory allocation in `MysqlIO.readPacket()` (bug 488663).

- Implementation of `DatabaseMetaData.getExported/ImportedKeys()` and `getCrossReference()`.
- Full synchronization on methods modifying instance and class-shared references, driver should be entirely thread-safe now (please let me know if you have problems).
- `DataSource` implementations moved to `org.gjt.mm.mysql.jdbc2.optional` package, and (initial) implementations of `PooledConnectionDataSource` and `XADataSource` are in place (thanks to Todd Wolff for the implementation and testing of `PooledConnectionDataSource` with IBM WebSphere 4).
- Added detection of network connection being closed when reading packets (thanks to Todd Lizambri).
- Fixed quoting error with escape processor (bug 486265).
- Report batch update support through `DatabaseMetaData` (bug 495101).
- Fixed off-by-one-hour error in `PreparedStatement.setTimestamp()` (bug 491577).
- Removed concatenation support from driver (the `||` operator), as older versions of VisualAge seem to be the only thing that use it, and it conflicts with the logical `||` operator. You will need to start **mysqld** with the `--ansi` flag to use the `||` operator as concatenation (bug 491680).
- Fixed casting bug in `PreparedStatement` (bug 488663).

D.5.43. Changes in MySQL Connector/J 2.0.8 (25 November 2001)

- Batch updates now supported (thanks to some inspiration from Daniel Rall).
- `XADataSource/ConnectionPoolDataSource` code (experimental)
- `PreparedStatement.setAnyNumericType()` now handles positive exponents correctly (adds `+` so MySQL can understand it).

- `DatabaseMetaData.getPrimaryKeys()` and `getBestRowIdentifier()` are now more robust in identifying primary keys (matches regardless of case or abbreviation/full spelling of Primary Key in Key_type column).

D.5.44. Changes in MySQL Connector/J 2.0.7 (24 October 2001)

- `PreparedStatement.setCharacterStream()` now implemented
- Fixed dangling socket problem when in high availability (`autoReconnect=true`) mode, and finalizer for `Connection` will close any dangling sockets on GC.
- Fixed `ResultSetMetaData.getPrecision()` returning one less than actual on newer versions of MySQL.
- `ResultSet.getBlob()` now returns `null` if column value was `null`.
- Character sets read from database if `useUnicode=true` and `characterEncoding` is not set. (thanks to Dmitry Vereshchagin)
- Initial transaction isolation level read from database (if available). (thanks to Dmitry Vereshchagin)
- Fixed `DatabaseMetaData.supportsTransactions()`, and `supportsTransactionIsolationLevel()` and `getTypeInfo()` `SQL_DATETIME_SUB` and `SQL_DATA_TYPE` fields not being readable.
- Fixed `PreparedStatement` generating SQL that would end up with syntax errors for some queries.
- Fixed `ResultSet.isAfterLast()` always returning `false`.
- Fixed time zone issue in `PreparedStatement.setTimestamp()`. (thanks to Erik Olofsson)
- Capitalize type names when `capitalizeTypeNames=true` is passed in URL or properties (for `WebObjects`). (thanks to Anjo Krank)
- Updatable result sets now correctly handle `NULL` values in fields.

- `PreparedStatement.setDouble()` now uses full-precision doubles (reverting a fix made earlier to truncate them).
- `PreparedStatement.setBoolean()` will use 1/0 for values if your MySQL version is 3.21.23 or higher.

D.5.45. Changes in MySQL Connector/J 2.0.6 (16 June 2001)

- Fixed `PreparedStatement` parameter checking.
- Fixed case-sensitive column names in `ResultSet.java`.

D.5.46. Changes in MySQL Connector/J 2.0.5 (13 June 2001)

- Fixed `ResultSet.getBlob()` `ArrayIndex` out-of-bounds.
- Fixed `ResultSetMetaData.getColumnTypeName` for TEXT/BLOB.
- Fixed `ArrayIndexOutOfBoundsException` when sending large BLOB queries. (Max size packet was not being set)
- Added ISOLATION level support to `Connection.setIsolationLevel()`
- Fixed NPE on `PreparedStatement.executeUpdate()` when all columns have not been set.
- Fixed data parsing of `TIMESTAMP` values with 2-digit years.
- Added `Byte` to `PreparedStatement.setObject()`.
- `ResultSet.getBoolean()` now recognizes -1 as true.
- `ResultSet` has +/-Inf/inf support.
- `ResultSet.insertRow()` works now, even if not all columns are set (they will be set to NULL).
- `DataBaseMetaData.getCrossReference()` no longer `ArrayIndex00B`.
- `getObject()` on `ResultSet` correctly does `TINYINT->Byte` and `SMALLINT-`

>Short.

D.5.47. Changes in MySQL Connector/J 2.0.3 (03 December 2000)

- Implemented `getBigDecimal()` without scale component for JDBC2.
- Fixed composite key problem with updatable result sets.
- Added detection of `-/+INF` for doubles.
- Faster ASCII string operations.
- Fixed incorrect detection of `MAX_ALLOWED_PACKET`, so sending large blobs should work now.
- Fixed off-by-one error in `java.sql.Blob` implementation code.
- Added `ultraDevHack` URL parameter, set to `true` to allow (broken) Macromedia UltraDev to use the driver.

D.5.48. Changes in MySQL Connector/J 2.0.1 (06 April 2000)

- Fixed `RSMD.isWritable()` returning wrong value. Thanks to Moritz Maass.
- Cleaned up exception handling when driver connects.
- Columns that are of type `TEXT` now return as `Strings` when you use `getObject()`.
- `DatabaseMetaData.getPrimaryKeys()` now works correctly with respect to `key_seq`. Thanks to Brian Slesinsky.
- No escape processing is done on `PreparedStatement`s anymore per JDBC spec.
- Fixed many JDBC-2.0 traversal, positioning bugs, especially with respect to empty result sets. Thanks to Ron Smits, Nick Brook, Cessar Garcia and Carlos Martinez.
- Fixed some issues with updatability support in `ResultSet` when using

multiple primary keys.

D.5.49. Changes in MySQL Connector/J 2.0.0pre5 (21 February 2000)

- Fixed Bad Handshake problem.

D.5.50. Changes in MySQL Connector/J 2.0.0pre4 (10 January 2000)

- Fixes to ResultSet for insertRow() - Thanks to Cesar Garcia
- Fix to Driver to recognize JDBC-2.0 by loading a JDBC-2.0 class, instead of relying on JDK version numbers. Thanks to John Baker.
- Fixed ResultSet to return correct row numbers
- Statement.getUpdateCount() now returns rows matched, instead of rows actually updated, which is more SQL-92 like.

10-29-99

- Statement/PreparedStatement.getMoreResults() bug fixed. Thanks to Noel J. Bergman.
- Added Short as a type to PreparedStatement.setObject(). Thanks to Jeff Crowder
- Driver now automagically configures maximum/preferred packet sizes by querying server.
- Autoreconnect code uses fast ping command if server supports it.
- Fixed various bugs with respect to packet sizing when reading from the server and when allocating to write to the server.

D.5.51. Changes in MySQL Connector/J 2.0.0pre (17 August 1999)

- Now compiles under JDK-1.2. The driver supports both JDK-1.1 and JDK-1.2 at the same time through a core set of classes. The driver will load the appropriate interface classes at runtime by figuring out which JVM version you are using.
- Fixes for result sets with all nulls in the first row. (Pointed out by Tim Endres)
- Fixes to column numbers in SQLExceptions in ResultSet (Thanks to Blas Rodriguez Somoza)
- The database no longer needs to be specified to connect. (Thanks to Christian Motschke)

D.5.52. Changes in MySQL Connector/J 1.2b (04 July 1999)

- Better Documentation (in progress), in doc/mm.doc/book1.html
- DBMD now allows null for a column name pattern (not in spec), which it changes to '%'
- DBMD now has correct types/lengths for getXXX().
- ResultSet.getDate(), getTime(), and getTimestamp() fixes. (contributed by Alan Wilken)
- EscapeProcessor now handles \{ \} and { or } inside quotes correctly. (thanks to Alik for some ideas on how to fix it)
- Fixes to properties handling in Connection. (contributed by Juho Tikkala)
- ResultSet.getObject() now returns null for NULL columns in the table, rather than bombing out. (thanks to Ben Grosman)
- ResultSet.getObject() now returns Strings for types from MySQL that it doesn't know about. (Suggested by Chris Perdue)
- Removed DataInput/Output streams, not needed, 1/2 number of method calls per IO operation.

- Use default character encoding if one is not specified. This is a work-around for broken JVMs, because according to spec, EVERY JVM must support "ISO8859_1", but they don't.
- Fixed Connection to use the platform character encoding instead of "ISO8859_1" if one isn't explicitly set. This fixes problems people were having loading the character- converter classes that didn't always exist (JVM bug). (thanks to Fritz Elfert for pointing out this problem)
- Changed MysqlIO to re-use packets where possible to reduce memory usage.
- Fixed escape-processor bugs pertaining to {} inside quotes.

D.5.53. Changes in MySQL Connector/J 1.2a (14 April 1999)

- Fixed character-set support for non-Javasoftware JVMs (thanks to many people for pointing it out)
- Fixed ResultSet.getBoolean() to recognize 'y' & 'n' as well as '1' & '0' as boolean flags. (thanks to Tim Pizey)
- Fixed ResultSet.getTimestamp() to give better performance. (thanks to Richard Swift)
- Fixed getByte() for numeric types. (thanks to Ray Bellis)
- Fixed DatabaseMetaData.getTypeInfo() for DATE type. (thanks to Paul Johnston)
- Fixed EscapeProcessor for "fn" calls. (thanks to Piyush Shah at locomotive.org)
- Fixed EscapeProcessor to not do extraneous work if there are no escape codes. (thanks to Ryan Gustafson)
- Fixed Driver to parse URLs of the form "jdbc:mysql://host:port" (thanks to Richard Lobb)

D.5.54. Changes in MySQL Connector/J 1.1i (24 March 1999)

- Fixed Timestamps for PreparedStatements
- Fixed null pointer exceptions in RSMD and RS
- Re-compiled with jikes for valid class files (thanks ms!)

D.5.55. Changes in MySQL Connector/J 1.1h (08 March 1999)

- Fixed escape processor to deal with unmatched { and } (thanks to Craig Coles)
- Fixed escape processor to create more portable (between DATETIME and TIMESTAMP types) representations so that it will work with BETWEEN clauses. (thanks to Craig Longman)
- MysqlIO.quit() now closes the socket connection. Before, after many failed connections some OS's would run out of file descriptors. (thanks to Michael Brinkman)
- Fixed NullPointerException in Driver.getPropertyInfo. (thanks to Dave Potts)
- Fixes to MysqlDefs to allow all *text fields to be retrieved as Strings. (thanks to Chris at Leverage)
- Fixed setDouble in PreparedStatement for large numbers to avoid sending scientific notation to the database. (thanks to J.S. Ferguson)
- Fixed getScale() and getPrecision() in RSMD. (contrib'd by James Klicman)
- Fixed getObject() when field was DECIMAL or NUMERIC (thanks to Bert Hobbs)
- DBMD.getTables() bombed when passed a null table-name pattern. Fixed. (thanks to Richard Lobb)
- Added check for "client not authorized" errors during connect. (thanks to Hannes Wallnoefer)

D.5.56. Changes in MySQL Connector/J 1.1g (19 February 1999)

- Result set rows are now byte arrays. Blobs and Unicode work bidirectionally now. The useUnicode and encoding options are implemented now.
- Fixes to PreparedStatement to send binary set by setXXXStream to be sent untouched to the MySQL server.
- Fixes to getDriverPropertyInfo().

D.5.57. Changes in MySQL Connector/J 1.1f (31 December 1998)

- Changed all ResultSet fields to Strings, this should allow Unicode to work, but your JVM must be able to convert between the character sets. This should also make reading data from the server be a bit quicker, because there is now no conversion from StringBuffer to String.
- Changed PreparedStatement.streamToString() to be more efficient (code from Uwe Schaefer).
- URL parsing is more robust (throws SQL exceptions on errors rather than NullPointerExceptions)
- PreparedStatement now can convert Strings to Time/Date values via setObject() (code from Robert Currey).
- IO no longer hangs in Buffer.readInt(), that bug was introduced in 1.1d when changing to all byte-arrays for result sets. (Pointed out by Samo Login)

D.5.58. Changes in MySQL Connector/J 1.1b (03 November 1998)

- Fixes to DatabaseMetaData to allow both IBM VA and J-Builder to work. Let me know how it goes. (thanks to Jac Kersing)
- Fix to ResultSet.getBoolean() for NULL strings (thanks to Barry Lagerweij)
- Beginning of code cleanup, and formatting. Getting ready to branch this off to a parallel JDBC-2.0 source tree.

- Added "final" modifier to critical sections in MysqlIO and Buffer to allow compiler to inline methods for speed.

9-29-98

- If object references passed to setXXX() in PreparedStatement are null, setNull() is automatically called for you. (Thanks for the suggestion goes to Erik Ostrom)
- setObject() in PreparedStatement will now attempt to write a serialized representation of the object to the database for objects of Types.OTHER and objects of unknown type.
- Util now has a static method readObject() which given a ResultSet and a column index will re-instantiate an object serialized in the above manner.

D.5.59. Changes in MySQL Connector/J 1.1 (02 September 1998)

- Got rid of "ugly hack" in MysqlIO.nextRow(). Rather than catch an exception, Buffer.isLastDataPacket() was fixed.
- Connection.getCatalog() and Connection.setCatalog() should work now.
- Statement.setMaxRows() works, as well as setting by property maxRows. Statement.setMaxRows() overrides maxRows set via properties or url parameters.
- Automatic re-connection is available. Because it has to "ping" the database before each query, it is turned off by default. To use it, pass in "autoReconnect=true" in the connection URL. You may also change the number of reconnect tries, and the initial timeout value via "maxReconnects=n" (default 3) and "initialTimeout=n" (seconds, default 2) parameters. The timeout is an exponential backoff type of timeout; for example, if you have initial timeout of 2 seconds, and maxReconnects of 3, then the driver will timeout 2 seconds, 4 seconds, then 16 seconds between each re-connection attempt.

D.5.60. Changes in MySQL Connector/J 1.0 (24 August 1998)

- Fixed handling of blob data in Buffer.java
- Fixed bug with authentication packet being sized too small.
- The JDBC Driver is now under the LPGL

8-14-98

- Fixed Buffer.readLenString() to correctly read data for BLOBS.
- Fixed PreparedStatement.stringToStream to correctly read data for BLOBS.
- Fixed PreparedStatement.setDate() to not add a day. (above fixes thanks to Vincent Partington)
- Added URL parameter parsing (?user=... and so forth).

D.5.61. Changes in MySQL Connector/J 0.9d (04 August 1998)

- Big news! New package name. Tim Endres from ICE Engineering is starting a new source tree for GNU GPL'd Java software. He's graciously given me the org.gjt.mm package directory to use, so now the driver is in the org.gjt.mm.mysql package scheme. I'm "legal" now. Look for more information on Tim's project soon.
- Now using dynamically sized packets to reduce memory usage when sending commands to the DB.
- Small fixes to getTypeInfo() for parameters, and so forth.
- DatabaseMetaData is now fully implemented. Let me know if these drivers work with the various IDEs out there. I've heard that they're working with JBuilder right now.
- Added JavaDoc documentation to the package.
- Package now available in .zip or .tar.gz.

D.5.62. Changes in MySQL Connector/J 0.9 (28 July 1998)

- Implemented `getTypeInfo()`. `Connection.rollback()` now throws an `SQLException` per the JDBC spec.
- Added `PreparedStatement` that supports all JDBC API methods for `PreparedStatement` including `InputStreams`. Please check this out and let me know if anything is broken.
- Fixed a bug in `ResultSet` that would break some queries that only returned 1 row.
- Fixed bugs in `DatabaseMetaData.getTables()`, `DatabaseMetaData.getColumns()` and `DatabaseMetaData.getCatalogs()`.
- Added functionality to `Statement` that allows `executeUpdate()` to store values for IDs that are automatically generated for `AUTO_INCREMENT` fields. Basically, after an `executeUpdate()`, look at the `SQLWarnings` for warnings like `"LAST_INSERTED_ID = 'some number', COMMAND = 'your SQL query'"`. If you are using `AUTO_INCREMENT` fields in your tables and are executing a lot of `executeUpdate()`s on one `Statement`, be sure to `clearWarnings()` every so often to save memory.

D.5.63. Changes in MySQL Connector/J 0.8 (06 July 1998)

- Split `MysqlIO` and `Buffer` to separate classes. Some `ClassLoaders` gave an `IllegalAccessException` error for some fields in those two classes. Now `mm.mysql` works in applets and all classloaders. Thanks to Joe Ennis <jce@mail.boone.com> for pointing out the problem and working on a fix with me.

D.5.64. Changes in MySQL Connector/J 0.7 (01 July 1998)

- Fixed `DatabaseMetadata` problems in `getColumns()` and bug in `switch` statement in the `Field` constructor. Thanks to Costin Manolache <costin@tdiinc.com> for pointing these out.

D.5.65. Changes in MySQL Connector/J 0.6 (21 May 1998)

- Incorporated efficiency changes from Richard Swift <Richard.Swift@kanatek.ca> in `MysqlIO.java` and `ResultSet.java`:

- We're now 15% faster than gwe's driver.
- Started working on DatabaseMetaData.
- The following methods are implemented:
 - `getTables()`
 - `getTableTypes()`
 - `getColumns`
 - `getCatalogs()`

Appendix E. Porting to Other Systems

Table of Contents

[E.1. Debugging a MySQL Server](#)

[E.1.1. Compiling MySQL for Debugging](#)

[E.1.2. Creating Trace Files](#)

[E.1.3. Debugging **mysqld** under **gdb**](#)

[E.1.4. Using a Stack Trace](#)

[E.1.5. Using Server Logs to Find Causes of Errors in **mysqld**](#)

[E.1.6. Making a Test Case If You Experience Table Corruption](#)

[E.2. Debugging a MySQL Client](#)

[E.3. The DBUG Package](#)

[E.4. Comments about RTS Threads](#)

[E.5. Differences Between Thread Packages](#)

This appendix helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See [Section 2.1.1, “Operating Systems Supported by MySQL”](#). If you have created a new port of MySQL, please let us know so that we can list it here and on our Web site (<http://www.mysql.com/>), recommending it to other users.

Note: If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copyright holder of MySQL.

A working POSIX thread library is needed for the server. On Solaris 2.5 we use Sun PThreads (the native thread support in 2.4 and earlier versions is not good enough), on Linux we use LinuxThreads by Xavier Leroy, <Xavier.Leroy@inria.fr>.

The hard part of porting to a new Unix variant without good native thread support is probably to port MIT-pthreads. See [mit-pthreads/README](#) and Programming POSIX Threads (<http://www.humanfactor.com/pthreads/>).

Up to MySQL 4.0.2, the MySQL distribution included a patched version of Chris Provenzano's Pthreads from MIT (see the MIT Pthreads Web page at <http://www.mit.edu/afs/sipb/project/pthreads/> and a programming introduction at http://www.mit.edu:8001/people/proven/IAP_2000/). These can be used for

some operating systems that do not have POSIX threads. See [Section 2.9.5, “MIT-pthreads Notes”](#).

It is also possible to use another user level thread package named FSU Pthreads (see <http://moss.csc.ncsu.edu/~mueller/pthreads/>). This implementation is being used for the SCO port.

See the `thr_lock.c` and `thr_alarm.c` programs in the `mysys` directory for some tests/examples of these problems.

Both the server and the client need a working C++ compiler. We use **gcc** on many platforms. Other compilers that are known to work are SPARCworks, Sun Forte, Irix **cc**, HP-UX **aCC**, IBM AIX **xlC_r**), Intel **ecc/icc** and Compaq **cxx**).

To compile only the client use **`./configure --without-server`**.

There is currently no support for only compiling the server, nor is it likely to be added unless someone has a good reason for it.

If you want/need to change any `Makefile` or the `configure` script you also need GNU Automake and Autoconf. See [Section 2.9.3, “Installing from the Development Source Tree”](#).

All steps needed to remake everything from the most basic files.

```
/bin/rm */.deps/*.P
/bin/rm -f config.cache
aclocal
autoheader
aclocal
automake
autoconf
./configure --with-debug=full --prefix='your installation directory'

# The makefiles generated above need GNU make 3.75 or newer.
# (called gmake below)
gmake clean all install init-db
```

If you run into problems with a new port, you may have to do some debugging of MySQL! See [Section E.1, “Debugging a MySQL Server”](#).

Note: Before you start debugging **mysqld**, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This ensures that your thread

installation has even a remote chance to work!

E.1. Debugging a MySQL Server

If you are using some functionality that is very new in MySQL, you can try to run **mysqld** with the `--skip-new` (which disables all new, potentially unsafe functionality) or with `--safe-mode` which disables a lot of optimization that may cause problems. See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#).

If **mysqld** doesn't want to start, you should verify that you don't have any `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with **mysqld --print-defaults** and avoid using them by starting with **mysqld --no-defaults ...**

If **mysqld** starts to eat up CPU or memory or if it “hangs,” you can use **mysqladmin processlist status** to find out if someone is executing a query that takes a long time. It may be a good idea to run **mysqladmin -i10 processlist status** in some window if you are experiencing performance problems or problems when new clients can't connect.

The command **mysqladmin debug** dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or **myisamchk**. See [Chapter 5, *Database Administration*](#). You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See [Section 2.13, “Operating System-Specific Notes”](#).

E.1.1. Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `--with-debug` or the `--with-debug=full` option. You can check whether MySQL was compiled with debugging by doing: **mysqld --help**. If the `--debug` flag is listed with the options then you have debugging enabled. **mysqladmin ver** also lists the **mysqld**

version as **mysql ... --debug** in this case.

If you are using **gcc** or **egcs**, the recommended **configure** line is:

```
CC=gcc CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors \  
-fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \  
--with-debug --with-extra-charsets=complex
```

This avoids problems with the `libstdc++` library and with C++ exceptions (many compilers have problems with C++ exceptions in threaded code) and compile a MySQL version with support for all character sets.

If you suspect a memory overrun error, you can configure MySQL with `--with-debug=full`, which installs a memory allocation (`SAFEMALLOC`) checker. However, running with `SAFEMALLOC` is quite slow, so if you get performance problems you should start **mysqld** with the `--skip-safemalloc` option. This disables the memory overrun checks for each call to `malloc()` and `free()`.

If **mysqld** stops crashing when you compile it with `--with-debug`, you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add `-g` to the `CFLAGS` and `CXXFLAGS` variables above and not use `--with-debug`. If **mysqld** dies, you can at least attach to it with **gdb** or use **gdb** on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of **mysqld**. If they find something “unexpected,” an entry is written to `stderr`, which **mysqld_safe** directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging! (The second thing is to send mail to a MySQL mailing list and ask for help. See [Section 1.7.1, “MySQL Mailing Lists”](#). If you believe that you have found a bug, please use the instructions at [Section 1.8, “How to Report Bugs or Problems”](#)).

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

E.1.2. Creating Trace Files

If the **mysqld** server doesn't start or if you can cause it to crash quickly, you can

try to create a trace file to find the problem.

To do this, you must have a **mysqld** that has been compiled with debugging support. You can check this by executing `mysqld -v`. If the version number ends with `-debug`, it's compiled with support for trace files. (On Windows, the debugging server is named **mysqld-debug** rather than **mysqld** as of MySQL 4.1.)

Start the **mysqld** server with a trace log in `/tmp/mysqld.trace` on Unix or `C:\mysqld.trace` on Windows:

```
shell> mysqld --debug
```

On Windows, you should also use the `--standalone` flag to not start **mysqld** as a service. In a console window, use this command:

```
C:\> mysqld-debug --debug --standalone
```

After this, you can use the `mysql.exe` command-line tool in a second console window to reproduce the problem. You can stop the **mysqld** server with **mysqladmin shutdown**.

Note that the trace file become **very big!** If you want to generate a smaller trace file, you can use debugging options something like this:

```
mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace
```

This only prints information with the most interesting tags to the trace file.

If you make a bug report about this, please only send the lines from the trace file to the appropriate mailing list where something seems to go wrong! If you can't locate the wrong place, you can ftp the trace file, together with a full bug report, to <ftp://ftp.mysql.com/pub/mysql/upload/> so that a MySQL developer can take a look at it.

The trace file is made with the **DEBUG** package by Fred Fish. See [Section E.3, "The DEBUG Package"](#).

E.1.3. Debugging mysqld under gdb

On most systems you can also start **mysqld** from **gdb** to get more information if **mysqld** crashes.

With some older **gdb** versions on Linux you must use `run --one-thread` if you want to be able to debug **mysqld** threads. In this case, you can only have one thread active at a time. We recommend you to upgrade to **gdb** 5.1 ASAP as thread debugging works much better with this version!

NTPL threads (the new thread library on Linux) may cause problems while running **mysqld** under **gdb**. Some symptoms are:

- **mysqld** hangs during startup (before it writes ready for connections).
- **mysqld** crashes during a `pthread_mutex_lock()` or `pthread_mutex_unlock()` call.

In this case, you should set the following environment variable in the shell before starting **gdb**:

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

When running **mysqld** under **gdb**, you should disable the stack trace with `--skip-stack-trace` to be able to catch segfaults within **gdb**.

In MySQL 4.0.14 and above you should use the `--gdb` option to **mysqld**. This installs an interrupt handler for `SIGINT` (needed to stop **mysqld** with `^C` to set breakpoints) and disable stack tracing and core file handling.

It's very hard to debug MySQL under **gdb** if you do a lot of new connections the whole time as **gdb** doesn't free the memory for old threads. You can avoid this problem by starting **mysqld** with `--thread_cache_size='max_connections+1'`. In most cases just using `--thread_cache_size=5` helps a lot!

If you want to get a core dump on Linux if **mysqld** dies with a `SIGSEGV` signal, you can start **mysqld** with the `--core-file` option. This core file can be used to make a backtrace that may help you find out why **mysqld** died:

```
shell> gdb mysqld core
gdb> backtrace full
gdb> quit
```

See [Section A.4.2, “What to Do If MySQL Keeps Crashing”](#).

If you are using **gdb** 4.17.x or above on Linux, you should install a `.gdb` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with **gdb**, you should download `gdb 5.x` and try this instead. The new **gdb** version has very improved thread handling!

Here is an example how to debug `mysqld`:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

Include the above output in a bug report, which you can file using the instructions in [Section 1.8, “How to Report Bugs or Problems”](#).

If **mysqld** hangs you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where **mysqld** has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl DBI interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable.

E.1.4. Using a Stack Trace

On some operating systems, the error log contains a stack trace if **mysqld** dies unexpectedly. You can use this to find out where (and maybe why) **mysqld** died. See [Section 5.12.1, “The Error Log”](#). To get a stack trace, you must not compile **mysqld** with the `-fomit-frame-pointer` option to `gcc`. See [Section E.1.1, “Compiling MySQL for Debugging”](#).

If the error file contains something like the following:

```
mysqld got signal 11;
The manual section 'Debugging a MySQL server' tells you how to use a
stack trace and/or the core file to produce a readable backtrace that
helps in finding out why mysqld died
Attempting backtrace. You can use the following information to find
where mysqld died. If you see no messages after this, something went
terribly wrong...
stack range sanity check, ok, backtrace follows
0x40077552
0x81281a0
0x8128f47
0x8127be0
0x8127995
0x8104947
0x80ff28f
0x810131b
0x80ee4bc
0x80c3c91
0x80c6b43
0x80c1fd9
0x80c1686
```

you can find where **mysqld** died by doing the following:

1. Copy the preceding numbers to a file, for example `mysqld.stack`.
2. Make a symbol file for the **mysqld** server:

```
nm -n libexec/mysqld > /tmp/mysqld.sym
```

Note that most MySQL binary distributions (except for the "debug" packages, where this information is included inside of the binaries themselves) ship with the above file, named `mysqld.sym.gz`. In this case, you can simply unpack it by doing:

```
gunzip < bin/mysqld.sym.gz > /tmp/mysqld.sym
```

3. Execute `resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack`.

This prints out where **mysqld** died. If this doesn't help you find out why **mysqld** died, you should make a bug report and include the output from the above command with the bug report.

Note however that in most cases it does not help us to just have a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, we would in most cases need to know the query that killed **mysqld** and preferably a test case so that we can repeat the problem! See [Section 1.8, “How to Report Bugs or Problems”](#).

E.1.5. Using Server Logs to Find Causes of Errors in **mysqld**

Note that before starting **mysqld** with `--log` you should check all your tables with **myisamchk**. See [Chapter 5, Database Administration](#).

If **mysqld** dies or hangs, you should start **mysqld** with `--log`. When **mysqld** dies again, you can examine the end of the log file for the query that killed **mysqld**.

If you are using `--log` without a file name, the log is stored in the database directory as `host_name.log`. In most cases it is the last query in the log file that killed **mysqld**, but if possible you should verify this by restarting **mysqld** and executing the found query from the **mysql** command-line tools. If this works, you should also test all complicated queries that didn't complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that **mysqld** is using indexes properly. See [Section 7.2.1, “Optimizing Queries with EXPLAIN”](#).

You can find the queries that take a long time to execute by starting **mysqld** with `--log-slow-queries`. See [Section 5.12.4, “The Slow Query Log”](#).

If you find the text `mysqld restarted` in the error log file (normally named `hostname.err`) you probably have found a query that causes **mysqld** to fail. If this happens, you should check all your tables with **myisamchk** (see [Chapter 5, Database Administration](#)), and test the queries in the MySQL log files to see whether one fails. If you find such a query, try first upgrading to the newest MySQL version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to a MySQL mailing list. The mailing lists are described at <http://lists.mysql.com/>, which also has links to online list archives.

If you have started **mysqld** with `myisam-recover`, MySQL automatically checks

and tries to repair MyISAM tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file 'Warning: Checking table ...' which is followed by warning: Repairing table if the table needs to be repaired. If you get a lot of these errors, without **mysqld** having died unexpectedly just before, then something is wrong and needs to be investigated further. See [Section 5.2.1, “mysqld Command Options”](#).

It is not a good sign if **mysqld** did die unexpectedly, but in this case, you should not investigate the Checking table... messages, but instead try to find out why **mysqld** died.

E.1.6. Making a Test Case If You Experience Table Corruption

If you get corrupted tables or if **mysqld** always fails after some update commands, you can test whether this bug is reproducible by doing the following:

- Take down the MySQL daemon (with **mysqladmin shutdown**).
- Make a backup of the tables (to guard against the very unlikely case that the repair does something bad).
- Check all tables with **myisamchk -s database/*.MYI**. Repair any wrong tables with **myisamchk -r database/table.MYI**.
- Make a second backup of the tables.
- Remove (or move away) any old log files from the MySQL data directory if you need more space.
- Start **mysqld** with `--log-bin`. See [Section 5.12.3, “The Binary Log”](#). If you want to find a query that crashes **mysqld**, you should use `--log --log-bin`.
- When you have gotten a crashed table, stop the **mysqld** server.
- Restore the backup.
- Restart the **mysqld** server **without** `--log-bin`

- Re-execute the commands with **mysqlbinlog update-log-file | mysql**. The update log is saved in the MySQL database directory with the name `hostname-bin.#`.
- If the tables are corrupted again or you can get **mysqld** to die with the above command, you have found reproducible bug that should be easy to fix! FTP the tables and the binary log to <ftp://ftp.mysql.com/pub/mysql/upload/> and report it in our bugs database using the instructions given in [Section 1.8, “How to Report Bugs or Problems”](#). (Please note that the `/pub/mysql/upload/` FTP directory is not listable, so you'll not see what you've uploaded in your FTP client.) If you are a support customer, you can use the MySQL Customer Support Center <https://support.mysql.com/> to alert the MySQL team about the problem and have it fixed as soon as possible.

You can also use the script `mysql_find_rows` to just execute some of the update statements if you want to narrow down the problem.

E.2. Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `--with-debug` or `--with-debug=full`. See [Section 2.9.2, “Typical configure Options”](#).

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:0,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running **mysql** in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:0,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See [Section 1.8, “How to Report Bugs or Problems”](#).

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

E.3. The DEBUG Package

The MySQL server and most MySQL clients are compiled with the DEBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is debugging. See [Section E.1.2, “Creating Trace Files”](#).

This section summarizes the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support. For more information about programming with the DEBUG package, see the DEBUG manual in the debug directory of MySQL source distributions. It's best to use a recent distribution for MySQL 5.0 to get the most updated DEBUG manual.

You use the debug package by invoking a program with the `--debug="..."` or the `-#...` option.

Most MySQL programs have a default debug string that is used if you don't specify an option to `--debug`. The default trace file is usually `/tmp/program_name.trace` on Unix and `\program_name.trace` on Windows.

The debug control string is a sequence of colon-separated fields as follows:

```
<field_1>:<field_2>:...:<field_N>
```

Each field consists of a mandatory flag character followed by an optional `' , '` and comma-separated list of modifiers:

```
flag[,modifier,modifier,...,modifier]
```

The currently recognized flag characters are:

Flag	Description
d	Enable output from <code>DEBUG_<N></code> macros for the current state. May be followed by a list of keywords which selects output only for the DEBUG macros with that keyword. An empty list of keywords implies output for all macros.
	Delay after each debugger output line. The argument is the number of

D	tenths of seconds to delay, subject to machine capabilities. For example, -#D, 20 specifies a delay of two seconds.
f	Limit debugging, tracing, and profiling to the list of named functions. Note that a null list disables all functions. The appropriate d or t flags must still be given; this flag only limits their actions if they are enabled.
F	Identify the source file name for each line of debug or trace output.
i	Identify the process with the PID or thread ID for each line of debug or trace output.
g	Enable profiling. Create a file called <code>debugmon.out</code> containing information that can be used to profile the program. May be followed by a list of keywords that select profiling only for the functions in that list. A null list implies that all functions are considered.
L	Identify the source file line number for each line of debug or trace output.
n	Print the current function nesting depth for each line of debug or trace output.
N	Number each line of debug output.
o	Redirect the debugger output stream to the specified file. The default output is <code>stderr</code> .
O	Like <code>o</code> , but the file is really flushed between each write. When needed, the file is closed and reopened between each write.
p	Limit debugger actions to specified processes. A process must be identified with the <code>DEBUG_PROCESS</code> macro and match one in the list for debugger actions to occur.
P	Print the current process name for each line of debug or trace output.
r	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
S	Do function <code>_sanity(_file_, _line_)</code> at each debugged function until <code>_sanity()</code> returns something that differs from 0. (Mostly used with <code>safemalloc</code> to find memory leaks)
t	Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option.

Some examples of debug control strings that might appear on a shell command

line (the `-#` is typically used to introduce a control string to an application program) are:

```
-#d:t  
-#d:f,main,subr1:F:L:t,20  
-#d,input,output,files:n  
-#d:t:i:0,\\mysqld.trace
```

In MySQL, common tags to print (with the `d` option) are `enter`, `exit`, `error`, `warning`, `info`, and `loop`.

E.4. Comments about RTS Threads

I have tried to use the RTS thread packages with MySQL but stumbled on the following problems:

They use old versions of many POSIX calls and it is very tedious to make wrappers for all functions. I am inclined to think that it would be easier to change the thread libraries to the newest POSIX specification.

Some wrappers are currently written. See `mysys/my_pthread.c` for more info.

At least the following should be changed:

`pthread_get_specific` should use one argument. `sigwait` should take two arguments. A lot of functions (at least `pthread_cond_wait`, `pthread_cond_timedwait()`) should return the error code on error. Now they return -1 and set `errno`.

Another problem is that user-level threads use the ALRM signal and this aborts a lot of functions (`read`, `write`, `open`...). MySQL should do a retry on interrupt on all of these but it is not that easy to verify it.

The biggest unsolved problem is the following:

To get thread-level alarms I changed `mysys/thr_alarm.c` to wait between alarms with `pthread_cond_timedwait()`, but this aborts with error `EINTR`. I tried to debug the thread library as to why this happens, but couldn't find any easy solution.

If someone wants to try MySQL with RTS threads I suggest the following:

- Change functions MySQL uses from the thread library to POSIX. This shouldn't take that long.
- Compile all libraries with the `-DHAVE_rts_threads`.
- Compile `thr_alarm`.
- If there are some small differences in the implementation, they may be

fixed by changing `my_pthread.h` and `my_pthread.c`.

- Run `thr_alarm`. If it runs without any “warning,” “error,” or aborted messages, you are on the right track. Here is a successful run on Solaris:

```
Main thread: 1
Thread 0 (5) started
Thread: 5 Waiting
process_alarm
Thread 1 (6) started
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 1 (1) sec
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 2 (2) sec
Thread: 6 Simulation of no alarm needed
Thread: 6 Slept for 0 (3) sec
Thread: 6 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 4 (4) sec
Thread: 6 Waiting
process_alarm
thread_alarm
Thread: 5 Slept for 10 (10) sec
Thread: 5 Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6 Slept for 5 (5) sec
Thread: 6 Waiting
process_alarm
process_alarm

...
thread_alarm
Thread: 5 Slept for 0 (1) sec
end
```

E.5. Differences Between Thread Packages

MySQL is very dependent on the thread package used. So when choosing a good platform for MySQL, the thread package is very important.

There are at least three types of thread packages:

- User threads in a single process. Thread switching is managed with alarms and the threads library manages all non-thread-safe functions with locks. Read, write and select operations are usually managed with a thread-specific select that switches to another thread if the running threads have to wait for data. If the user thread packages are integrated in the standard libs (FreeBSD and BSDI threads) the thread package requires less overhead than thread packages that have to map all unsafe calls (MIT-pthreads, FSU Pthreads and RTS threads). In some environments (for example, SCO), all system calls are thread-safe so the mapping can be done very easily (FSU Pthreads on SCO). Downside: All mapped calls take a little time and it's quite tricky to be able to handle all situations. There are usually also some system calls that are not handled by the thread package (like MIT-pthreads and sockets). Thread scheduling isn't always optimal.
- User threads in separate processes. Thread switching is done by the kernel and all data are shared between threads. The thread package manages the standard thread calls to allow sharing data between threads. LinuxThreads is using this method. Downside: Lots of processes. Thread creating is slow. If one thread dies the rest are usually left hanging and you must kill them all before restarting. Thread switching is somewhat expensive.
- Kernel threads. Thread switching is handled by the thread library or the kernel and is very fast. Everything is done in one process, but on some systems, **ps** may show the different threads. If one thread aborts, the whole process aborts. Most system calls are thread-safe and should require very little overhead. Solaris, HP-UX, AIX and OSF/1 have kernel threads.

In some systems kernel threads are managed by integrating user level threads in the system libraries. In such cases, the thread switching can only be done by the thread library and the kernel isn't really “thread aware.”

Appendix F. Environment Variables

This appendix lists all the environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Note that any options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables.

In many cases, it is preferable to use an option file instead of environment variables to modify the behavior of MySQL. See [Section 4.3.2, “Using Option Files”](#).

Variable	Description
CXX	The name of your C++ compiler (for running configure).
CC	The name of your C compiler (for running configure).
CFLAGS	Flags for your C compiler (for running configure).
CXXFLAGS	Flags for your C++ compiler (for running configure).
DBI_USER	The default username for Perl DBI.
DBI_TRACE	Trace options for Perl DBI.
HOME	The default path for the mysql history file is \$HOME/.mysql_history.
LD_RUN_PATH	Used to specify the location of libmysqlclient.so.
MYSQL_DEBUG	Debug trace options when debugging.
MYSQL_GROUP_SUFFIX	Option group suffix value (like specifying --defaults-group-suffix).
MYSQL_HISTFILE	The path to the mysql history file. If this variable is set, its value overrides the default for \$HOME/.mysql_history.
MYSQL_HOME	The path to the directory in which the server-specific my.cnf file resides (as of MySQL 5.0.3).
MYSQL_HOST	The default hostname used by the mysql command-line client.

MYSQL_PS1	The command prompt to use in the mysql command-line client.
MYSQL_PWD	The default password when connecting to mysqld . Note that using this is insecure. See Section 5.9.6, “Keeping Your Password Secure” .
MYSQL_TCP_PORT	The default TCP/IP port number.
MYSQL_UNIX_PORT	The default Unix socket filename; used for connections to localhost.
PATH	Used by the shell to find MySQL programs.
TMPDIR	The directory where temporary files are created.
TZ	This should be set to your local time zone. See Section A.4.6, “Time Zone Problems” .
UMASK_DIR	The user-directory creation mask when creating directories. Note that this is ANDed with UMASK.
UMASK	The user-file creation mask when creating files.
USER	The default username on Windows and NetWare used when connecting to mysqld .

Appendix G. Regular Expressions

A regular expression is a powerful way of specifying a pattern for a complex search.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. See [Appendix C, Credits](#). MySQL uses the extended version to support pattern-matching operations performed with the REGEXP operator in SQL statements. See [Section 3.3.4.7, “Pattern Matching”](#), and [Section 12.3.1, “String Comparison Functions”](#).

This appendix is a summary, with examples, of the special characters and constructs that can be used in MySQL for REGEXP operations. It does not contain all the details that can be found in Henry Spencer's `regex(7)` manual page. That manual page is included in MySQL source distributions, in the `regex.7` file under the `regex` directory.

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Non-trivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|word` matches either the string `hello` or the string `word`.

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression for the REGEXP operator may use any of the following special characters and constructs:

- `^`

Match the beginning of a string.

```
mysql> SELECT 'fo\nfo' REGEXP '^fo$';           -> 0
mysql> SELECT 'fofo' REGEXP '^fo';             -> 1
```

- \$

Match the end of a string.

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';      -> 1
mysql> SELECT 'fo\no' REGEXP '^fo$';        -> 0
```

- .

Match any character (including carriage return and newline).

```
mysql> SELECT 'fofo' REGEXP '^f.*$';        -> 1
mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';    -> 1
```

- a*

Match any sequence of zero or more a characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';        -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';     -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';        -> 1
```

- a+

Match any sequence of one or more a characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';        -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';        -> 0
```

- a?

Match either zero or one a character.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';        -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';       -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';     -> 0
```

- de|abc

Match either of the sequences de or abc.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';       -> 1
mysql> SELECT 'axe' REGEXP 'pi|apa';     -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';     -> 1
```

```
mysql> SELECT 'apa' REGEXP '^(pi|apa)$';      -> 1
mysql> SELECT 'pi' REGEXP '^(pi|apa)$';      -> 1
mysql> SELECT 'pix' REGEXP '^(pi|apa)$';     -> 0
```

- (abc)*

Match zero or more instances of the sequence abc.

```
mysql> SELECT 'pi' REGEXP '^(pi)*$';         -> 1
mysql> SELECT 'pip' REGEXP '^(pi)*$';        -> 0
mysql> SELECT 'pipi' REGEXP '^(pi)*$';       -> 1
```

- {1}, {2, 3}

{n} or {m, n} notation provides a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. m and n are integers.

- a*

Can be written as a{0, }.

- a+

Can be written as a{1, }.

- a?

Can be written as a{0, 1}.

To be more precise, a{n} matches exactly n instances of a. a{n, } matches n or more instances of a. a{m, n} matches m through n instances of a, inclusive.

m and n must be in the range from 0 to RE_DUP_MAX (default 255), inclusive. If both m and n are given, m must be less than or equal to n.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';   -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';   -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e'; -> 1
```

- [a-dX], [^a-dX]

Matches any character that is (or is not, if ^ is used) either a, b, c, d or x. A - character between two other characters forms a range that matches all characters from the first character to the second. For example, [0-9] matches any decimal digit. To include a literal] character, it must immediately follow the opening bracket [. To include a literal - character, it must be written first or last. Any character that does not have a defined special meaning inside a [] pair matches only itself.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';           -> 1
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$';       -> 0
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]+$';      -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';     -> 0
mysql> SELECT 'gheis' REGEXP '^[^a-dXYZ]+$';    -> 1
mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$';   -> 0
```

- [.characters.]

Within a bracket expression (written using [and]), matches the sequence of characters of that collating element. characters is either a single character or a character name like `newline`. You can find the full list of character names in the `regex/cname.h` file.

```
mysql> SELECT '~' REGEXP '[[.~.]]';             -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';        -> 1
```

- [=character_class=]

Within a bracket expression (written using [and]), [=character_class=] represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if o and (+) are the members of an equivalence class, then [[=o=]], [[=(+)=]], and [o(+)] are all synonymous. An equivalence class may not be used as an endpoint of a range.

- [:character_class:]

Within a bracket expression (written using [and]), [:character_class:] represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the `ctype(3)` manual page. A particular locale may provide other class names. A character class may not be used as

an endpoint of a range.

alnum	Alphanumeric characters
alpha	Alphabetic characters
blank	Whitespace characters
cntrl	Control characters
digit	Digit characters
graph	Graphic characters
lower	Lowercase alphabetic characters
print	Graphic or space characters
punct	Punctuation characters
space	Space, tab, newline, and carriage return
upper	Uppercase alphabetic characters
xdigit	Hexadecimal digit characters

```
mysql> SELECT 'justalnums' REGEXP '[:alnum:]+';      -> 1
mysql> SELECT '!!' REGEXP '[:alnum:]+';             -> 0
```

- [[:<:]], [[:>:]]

These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the `alnum` class or an underscore (`_`).

```
mysql> SELECT 'a word a' REGEXP '[:<:]word[[:>:]]'; -> 1
mysql> SELECT 'a xword a' REGEXP '[:<:]word[[:>:]]'; -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (`\`) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special `+` character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT '1+2' REGEXP '1+2';                   -> 0
mysql> SELECT '1+2' REGEXP '1\+2';                  -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';                  -> 1
```

Appendix H. Limits in MySQL

Table of Contents

[H.1. Limits of Joins](#)

This Appendix lists current limits in MySQL 5.0.

H.1. Limits of Joins

The maximum number of tables that can be referenced in a single join is 61. This also applies to the number of tables that can be referenced in the definition of a view.

Appendix I. Feature Restrictions

Table of Contents

[I.1. Restrictions on Stored Routines and Triggers](#)

[I.2. Restrictions on Server-Side Cursors](#)

[I.3. Restrictions on Subqueries](#)

[I.4. Restrictions on Views](#)

[I.5. Restrictions on XA Transactions](#)

The discussion here describes restrictions that apply to the use of MySQL features such as subqueries or views.

I.1. Restrictions on Stored Routines and Triggers

Some of the restrictions noted here apply to all stored routines; that is, both to stored procedures and stored functions. Some of restrictions apply only to stored functions, and not to stored procedures.

All of the restrictions for stored functions also apply to triggers.

Stored routines cannot contain arbitrary SQL statements. The following statements are disallowed:

- The table-maintenance statements `CHECK TABLES` and `OPTIMIZE TABLES`.
Note: This restriction is lifted beginning with MySQL 5.0.17.
- The locking statements `LOCK TABLES`, `UNLOCK TABLES`.
- `LOAD DATA` and `LOAD TABLE`.
- SQL prepared statements (`PREPARE`, `EXECUTE`, `DEALLOCATE PREPARE`).
Implication: You cannot use dynamic SQL within stored routines (where you construct dynamically statements as strings and then execute them). This restriction is lifted as of MySQL 5.0.13 for stored procedures; it still applies to stored functions and triggers.

For stored functions (but not stored procedures), the following additional statements or operations are disallowed:

- Statements that do explicit or implicit commit or rollback.
- Statements that return a result set. This includes `SELECT` statements that do not have an `INTO var_list` clause and `SHOW` statements. A function can process a result set either with `SELECT ... INTO var_list` or by using a cursor and `FETCH` statements. See [Section 17.2.7.3, “SELECT ... INTO Statement”](#).
- `FLUSH` statements.
- **Note:** Before MySQL 5.0.10, stored functions created with `CREATE FUNCTION` must not contain references to tables, with limited exceptions.

They may include some SET statements that contain table references, for example `SET a:= (SELECT MAX(id) FROM t)`, and SELECT statements that fetch values directly into variables, for example `SELECT i INTO var1 FROM t`.

- Recursive statements. That is, stored functions cannot be used recursively.
- Within a stored function or trigger, it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

Note that although some restrictions normally apply to stored functions and triggers but not to stored procedures, those restrictions do apply to stored procedures if they are invoked from within a stored function or trigger. For example, although you can use `FLUSH` in a stored procedure, such a stored procedure cannot be called from a stored function or trigger.

It is possible for the same identifier to be used for a routine parameter, a local variable, and a table column. Also, the same local variable name can be used in nested blocks. For example:

```
CREATE PROCEDURE p (i INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  SELECT i FROM t;
  BEGIN
    DECLARE i INT DEFAULT 1;
    SELECT i FROM t;
  END;
END;
```

In such cases the identifier is ambiguous and the following precedence rules apply:

- A local variable takes precedence over a routine parameter or table column
- A routine parameter takes precedence over a table column
- A local variable in an inner block takes precedence over a local variable in an outer block

The behavior that table columns do not take precedence over variables is non-

standard.

Use of stored routines can cause replication problems. This issue is discussed further in [Section 17.4, “Binary Logging of Stored Routines and Triggers”](#).

INFORMATION_SCHEMA does not yet have a PARAMETERS table, so applications that need to acquire routine parameter information at runtime must use workarounds such as parsing the output of SHOW CREATE statements.

There are no stored routine debugging facilities.

CALL statements cannot be prepared. This true both for server-side prepared statements and for SQL prepared statements.

UNDO handlers are not supported.

FOR loops are not supported.

To prevent problems of interaction between server threads, when a client issues a statement, the server uses a snapshot of routines and triggers available for execution of the statement. That is, the server calculates a list of procedures, functions, and triggers that may be used during execution of the statement, loads them, and then proceeds to execute the statement. This means that while the statement executes, it will not see changes to routines performed by other threads.

For triggers, the following additional statements or operations are disallowed:

- Triggers currently are not activated by foreign key actions.
- The RETURN statement is disallowed in triggers, which cannot return a value. To exit a trigger immediately, use the LEAVE statement.
- Triggers are not allowed on tables in the mysql database.

I.2. Restrictions on Server-Side Cursors

Server-side cursors are implemented beginning with the C API in MySQL 5.0.2 via the `mysql_stmt_attr_set()` function. A server-side cursor allows a result set to be generated on the server side, but not transferred to the client except for those rows that the client requests. For example, if a client executes a query but is only interested in the first row, the remaining rows are not transferred.

In MySQL, a server-side cursor is materialized into a temporary table. Initially, this is a MEMORY table, but is converted to a MyISAM table if its size reaches the value of the `max_heap_table_size` system variable. (Beginning with MySQL 5.0.14, the same temporary-table implementation also is used for cursors in stored routines.) One limitation of the implementation is that for a large result set, retrieving its rows through a cursor might be slow.

Cursors are read-only; you cannot use a cursor to update rows.

UPDATE WHERE CURRENT OF and DELETE WHERE CURRENT OF are not implemented, because updatable cursors are not supported.

Cursors are non-holdable (not held open after a commit).

Cursors are asensitive.

Cursors are non-scrollable.

Cursors are not named. The statement handler acts as the cursor ID.

You can have open only a single cursor per prepared statement. If you need several cursors, you must prepare several statements.

You cannot use a cursor for a statement that generates a result set if the statement is not supported in prepared mode. This includes statements such as CHECK TABLES, HANDLER READ, and SHOW BINLOG EVENTS.

I.3. Restrictions on Subqueries

- Known bug to be fixed later: If you compare a NULL value to a subquery using ALL, ANY, or SOME, and the subquery returns an empty result, the comparison might evaluate to the non-standard result of NULL rather than to TRUE or FALSE.
- A subquery's outer statement can be any one of: SELECT, INSERT, UPDATE, DELETE, SET, or DO.
- Subquery optimization for IN is not as effective as for the = operator or for IN(value_list) constructs.

A typical case for poor IN subquery performance is when the subquery returns a small number of rows but the outer query returns a large number of rows to be compared to the subquery result.

The problem is that, for a statement that uses an IN subquery, the optimizer rewrites it as a correlated subquery. Consider the following statement that uses an uncorrelated subquery:

```
SELECT ... FROM t1 WHERE t1.a IN (SELECT b FROM t2);
```

The optimizer rewrites the statement to a correlated subquery:

```
SELECT ... FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.b = t
```

If the inner and outer queries return M and N rows, respectively, the execution time becomes on the order of $O(M \times N)$, rather than $O(M+N)$ as it would be for an uncorrelated subquery.

An implication is that an IN subquery can be much slower than a query written using an IN(value_list) construct that lists the same values that the subquery would return.

- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms:

```
DELETE FROM t WHERE ... (SELECT ... FROM t ...);
UPDATE t ... WHERE col = (SELECT ... FROM t ...);
{INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
```

Exception: The preceding prohibition does not apply if you are using a subquery for the modified table in the FROM clause. Example:

```
UPDATE t ... WHERE col = (SELECT (SELECT ... FROM t...) AS _t ..
```

Here the prohibition does not apply because the result from a subquery in the FROM clause is stored as a temporary table, so the relevant rows in t have already been selected by the time the update to t takes place.

- Row comparison operations are only partially supported:
 - For `expr IN (subquery)`, `expr` can be an n -tuple (specified via row constructor syntax) and the subquery can return rows of n -tuples.
 - For `expr op {ALL|ANY|SOME} (subquery)`, `expr` must be a scalar value and the subquery must be a column subquery; it cannot return multiple-column rows.

In other words, for a subquery that returns rows of n -tuples, this is supported:

```
(val1, ..., valn) IN (subquery)
```

But this is not supported:

```
(val1, ..., valn) op {ALL|ANY|SOME} (subquery)
```

The reason for supporting row comparisons for IN but not for the others is that IN is implemented by rewriting it as a sequence of = comparisons and AND operations. This approach cannot be used for ALL, ANY, or SOME.

- Row constructors are not well optimized. The following two expressions are equivalent, but only the second can be optimized:

```
(col1, col2, ...) = (val1, val2, ...)
col1 = val1 AND col2 = val2 AND ...
```

- Subqueries in the FROM clause cannot be correlated subqueries. They are

materialized (executed to produce a result set) before evaluating the outer query, so they cannot be evaluated per row of the outer query.

- The optimizer is more mature for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as a join.

An exception occurs for the case where an IN subquery can be rewritten as a SELECT DISTINCT join. Example:

```
SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE
```

That statement can be rewritten as follows:

```
SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND
```

But in this case, the join requires an extra DISTINCT operation and is not more efficient than the subquery.

- Possible future optimization: MySQL does not rewrite the join order for subquery evaluation. In some cases, a subquery could be executed more efficiently if MySQL rewrote it as a join. This would give the optimizer a chance to choose between more execution plans. For example, it could decide whether to read one table or the other first.

Example:

```
SELECT a FROM outer_table AS ot  
WHERE a IN (SELECT a FROM inner_table AS it WHERE ot.b = it.b);
```

For that query, MySQL always scans `outer_table` first and then executes the subquery on `inner_table` for each row. If `outer_table` has a lot of rows and `inner_table` has few rows, the query probably will not be as fast as it could be.

The preceding query could be rewritten like this:

```
SELECT a FROM outer_table AS ot, inner_table AS it  
WHERE ot.a = it.a AND ot.b = it.b;
```

In this case, we can scan the small table (`inner_table`) and look up rows in `outer_table`, which will be fast if there is an index on (`ot.a, ot.b`).

- Possible future optimization: A correlated subquery is evaluated for each row of the outer query. A better approach is that if the outer row values do not change from the previous row, do not evaluate the subquery again. Instead, use its previous result.
- Possible future optimization: A subquery in the FROM clause is evaluated by materializing the result into a temporary table, and this table does not use indexes. This does not allow the use of indexes in comparison with other tables in the query, although that might be useful.
- Possible future optimization: If a subquery in the FROM clause resembles a view to which the merge algorithm can be applied, rewrite the query and apply the merge algorithm so that indexes can be used. The following statement contains such a subquery:

```
SELECT * FROM (SELECT * FROM t1 WHERE t1.t1_col) AS _t1, t2 WHERE
```

The statement can be rewritten as a join like this:

```
SELECT * FROM t1, t2 WHERE t1.t1_col AND t2.t2_col;
```

This type of rewriting would provide two benefits:

- It avoids the use of a temporary table for which no indexes can be used. In the rewritten query, the optimizer can use indexes on t1.
 - It gives the optimizer more freedom to choose between different execution plans. For example, rewriting the query as a join allows the optimizer to use t1 or t2 first.
- Possible future optimization: For IN, = ANY, <> ANY, = ALL, and <> ALL with non-correlated subqueries, use an in-memory hash for a result result or a temporary table with an index for larger results. Example:

```
SELECT a FROM big_table AS bt
WHERE non_key_field IN (SELECT non_key_field FROM table WHERE cc
```

In this case, we could create a temporary table:

```
CREATE TABLE t (key (non_key_field))
(SELECT non_key_field FROM table WHERE condition)
```

Then, for each row in `big_table`, do a key lookup in `t` based on `bt.non_key_field`.

I.4. Restrictions on Views

View processing is not optimized:

- It is not possible to create an index on a view.
- Indexes can be used for views processed using the merge algorithm. However, a view that is processed with the temptable algorithm is unable to take advantage of indexes on its underlying tables (although indexes can be used during generation of the temporary tables).

Subqueries cannot be used in the `FROM` clause of a view. This limitation will be lifted in the future.

There is a general principle that you cannot modify a table and select from the same table in a subquery. See [Section I.3, “Restrictions on Subqueries”](#).

The same principle also applies if you select from a view that selects from the table, if the view selects from the table in a subquery and the view is evaluated using the merge algorithm. Example:

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

If the view is evaluated using a temporary table, you *can* select from the table in the view subquery and still modify that table in the outer query. In this case the view will be stored in a temporary table and thus you are not really selecting from the table in a subquery and modifying it “at the same time.” (This is another reason you might wish to force MySQL to use the temptable algorithm by specifying `ALGORITHM = TEMPTABLE` in the view definition.)

You can use `DROP TABLE` or `ALTER TABLE` to drop or alter a table that is used in a view definition (which invalidates the view) and no warning results from the drop or alter operation. An error occurs later when the view is used.

A view definition is “frozen” by certain statements:

- If a statement prepared by PREPARE refers to a view, the view contents seen each time the statement is executed later will be the contents of the view at the time it was prepared. This is true even if the view definition is changed after the statement is prepared and before it is executed. Example:

```
CREATE VIEW v AS SELECT 1;
PREPARE s FROM 'SELECT * FROM v';
ALTER VIEW v AS SELECT 2;
EXECUTE s;
```

The result returned by the EXECUTE statement is 1, not 2.

- If a statement in a stored routine refers to a view, the view contents seen by the statement are its contents the first time that statement is executed. For example, this means that if the statement is executed in a loop, further iterations of the statement see the same view contents, even if the view definition is changed later in the loop. Example:

```
CREATE VIEW v AS SELECT 1;
delimiter //
CREATE PROCEDURE p ()
BEGIN
    DECLARE i INT DEFAULT 0;
    WHILE i < 5 DO
        SELECT * FROM v;
        SET i = i + 1;
        ALTER VIEW v AS SELECT 2;
    END WHILE;
END;
//
delimiter ;
CALL p();
```

When the procedure p() is called, the SELECT returns 1 each time through the loop, even though the view definition is changed within the loop.

With regard to view updatability, the overall goal for views is that if any view is theoretically updatable, it should be updatable in practice. This includes views that have UNION in their definition. Currently, not all views that are theoretically updatable can be updated. The initial view implementation was deliberately written this way to get usable, updatable views into MySQL as quickly as possible. Many theoretically updatable views can be updated now, but limitations still exist:

- Updatable views with subqueries anywhere other than in the `WHERE` clause. Some views that have subqueries in the `SELECT` list may be updatable.
- You cannot use `UPDATE` to update more than one underlying table of a view that is defined as a join.
- You cannot use `DELETE` to update a view that is defined as a join.

I.5. Restrictions on XA Transactions

XA transaction support is limited to the InnoDB storage engine.

The MySQL XA implementation is for “external XA,” where a MySQL server acts as a Resource Manager and client programs act as Transaction Managers. “Internal XA” is not implemented. This would allow individual storage engines within a MySQL server to act as RMs, and the server itself to act as a TM. Internal XA is required for handling XA transactions that involve more than one storage engine. The implementation of internal XA is incomplete because it requires that a storage engine support two-phase commit at the table handler level, and currently this is true only for InnoDB.

For XA START, the JOIN and RESUME clauses are not supported.

For XA END, the SUSPEND [FOR MIGRATE] clause is not supported.

The requirement that the *bqual* part of the *xid* value be different for each XA transaction within a global transaction is a limitation of the current MySQL XA implementation. It is not part of the XA specification.

If an XA transaction has reached the PREPARED state and the MySQL server is killed (for example, with **kill -9** on Unix) or shuts down abnormally, the transaction can be continued after the server restarts. However, if the client reconnects and commits the transaction, the transaction will be absent from the binary log even though it has been committed. This means the data and the binary log have gone out of synchrony. An implication is that XA cannot be used safely together with replication.

It is possible that the server will roll back a pending XA transaction, even one that has reached the PREPARED state. This happens if a client connection terminates and the server continues to run, or if clients are connected and the server shuts down gracefully. (In the latter case, the server marks each connection to be terminated, and then rolls back the PREPARED XA transaction associated with it.) It should be possible to commit or roll back a PREPARED XA transaction, but this cannot be done without changes to the binary logging mechanism.

Appendix J. GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software---to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The ``Program'', below, refers to any such program or work, and a ``work based on the Program'' means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term ``modification''.) Each licensee is addressed as ``you''.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code

as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose

permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 1. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 2. Accompany it with a written offer, valid for at least three years, to give any third-party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source

code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other

pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and ``any later version'', you have the option of following the terms and conditions

either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM ``AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER

PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the ``copyright" line and a pointer to where the full notice is found.

one line to give the program's name and a brief idea of what it does
Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1300

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `sho
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate

parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items---whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a ``copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hax

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix K. MySQL FLOSS License Exception

The MySQL AB Exception for Free/Libre and Open Source Software-only Applications Using MySQL Client Libraries (the “FLOSS Exception”).

Version 0.4, 08 September 2005

Exception Intent

We want specified Free/Libre and Open Source Software (“FLOSS”) applications to be able to use specified GPL-licensed MySQL client libraries (the “Program”) despite the fact that not all FLOSS licenses are compatible with version 2 of the GNU General Public License (the “GPL”).

Legal Terms and Conditions

As a special exception to the terms and conditions of version 2.0 of the GPL:

1. You are free to distribute a Derivative Work that is formed entirely from the Program and one or more works (each, a “FLOSS Work”) licensed under one or more of the licenses listed below in section 1, as long as:
 1. You obey the GPL in all respects for the Program and the Derivative Work, except for identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves,
 2. all identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves,
 1. are distributed subject to one of the FLOSS licenses listed below, and
 2. the object code or executable form of those sections are accompanied by the complete corresponding machine-readable source code for those sections on the same medium and under the same FLOSS license as the corresponding object code or

executable forms of those sections, and

3. any works which are aggregated with the Program or with a Derivative Work on a volume of a storage or distribution medium in accordance with the GPL, can reasonably be considered independent and separate works in themselves which are not derivatives of either the Program, a Derivative Work or a FLOSS Work.

If the above conditions are not met, then the Program may only be copied, modified, distributed or used under the terms and conditions of the GPL or another valid licensing option from MySQL AB.

2. FLOSS License List

License name	Version(s)/Copyright Date
Academic Free License	2.0
Apache Software License	1.0/1.1/2.0
Apple Public Source License	2.0
Artistic license	From Perl 5.8.0
BSD license	"July 22 1999"
Common Public License	1.0
GNU Library or "Lesser" General Public License (LGPL)	2.0/2.1
Jabber Open Source License	1.0
MIT license	-
Mozilla Public License (MPL)	1.0/1.1
Open Software License	2.0
OpenSSL license (with original SSLeay license)	"2003" ("1998")
PHP License	3.0
Python license (CNRI Python License)	—
Python Software Foundation License	2.1.1
Sleepycat License	"1999"
W3C License	"2001"

X11 License	"2001"
Zlib/libpng License	—
Zope Public License	2.0

Due to the many variants of some of the above licenses, we require that any version follow the 2003 version of the Free Software Foundation's Free Software Definition (<http://www.gnu.org/philosophy/free-sw.html>) or version 1.9 of the Open Source Definition by the Open Source Initiative (<http://www.opensource.org/docs/definition.php>).

3. Definitions

1. Terms used, but not defined, herein shall have the meaning provided in the GPL.
2. Derivative Work means a derivative work under copyright law.
4. **Applicability:** This FLOSS Exception applies to all Programs that contain a notice placed by MySQL AB saying that the Program may be distributed under the terms of this FLOSS Exception. If you create or distribute a work which is a Derivative Work of both the Program and any other work licensed under the GPL, then this FLOSS Exception is not available for that work; thus, you must remove the FLOSS Exception notice from that work and comply with the GPL in all respects, including by retaining all GPL notices. You may choose to redistribute a copy of the Program exclusively under the terms of the GPL by removing the FLOSS Exception notice from that copy of the Program, provided that the copy has never been modified by you or any third party.

Appendix A. Qualified Libraries and Packages

The following is a non-exhaustive list of libraries and packages which are covered by the FLOSS License Exception. Please note that this appendix is provided merely as an additional service to specific FLOSS projects wishing to simplify licensing information for their users. Compliance with one of the licenses noted under the “FLOSS license list” section remains a prerequisite.

Package Name	Qualifying License and Version

||Apache Portable Runtime (APR)||Apache Software License 2.0 ||